Freie Universität Berlin

Bachelorarbeit am Institut für Informatik der Freien Universität Berlin

Arbeitsgruppe Software Engineering

Datenbankanwendung zu seltenen Haustierkrankheiten

Robert Businski
Matrikelnummer: 4480066
robertb84@zedat.fu-berlin.de

Betreuer:	Prof. Dr. Lutz Prechelt
Eingereicht bei:	Prof. Dr. Lutz Prechelt
Zweitgutachter:	Prof. Dr. Claudia Müller-Birn

Berlin, 2. August 2017

Zusammenfassung

Diese Bachelorarbeit beschreibt den iterativen Entwicklungsprozess einer Webanwendung und ihrer zugehörigen Datenbank.

Eidesstattliche Erklärung

Ich versichere hiermit an Eides Statt, dass diese Arbeit von niemand anderem als meiner Person verfasst worden ist. Alle verwendeten Hilfsmittel wie Berichte, Bücher, Internetseiten oder ähnliches sind im Literaturverzeichnis angegeben, Zitate aus fremden Arbeiten sind als solche kenntlich gemacht. Die Arbeit wurde bisher in gleicher oder ähnlicher Form keiner anderen Prüfungskommission vorgelegt und auch nicht veröffentlicht.

2. August 2017

Robert Businski

Inhaltsverzeichnis

1 Einleitung **1**
1.1 Motivation . . . 1
1.2 Ziel . . . 1
1.3 Kapitelübersicht . . . 1

2 Vorbereitungen **3**
2.1 Anforderungen . . . 3
2.1.1 Funktionale Anforderungen . . . 3
2.1.2 Nichtfunktionale Anforderungen . . . 3
2.2 Auswahl der Sprache . . . 4
2.3 Auswahl des Datenbankverwaltungssystems und der Engine . . . 4
2.4 Auswahl des Frameworks . . . 4
2.4.1 CakePHP . . . 4
2.4.2 Codeigniter . . . 5
2.4.3 Laravel . . . 5
2.5 Erstes Datenbankdesign . . . 5

3 Ausführung **8**
3.1 Grundstein . . . 8
3.2 Erste Iteration . . . 10
3.2.1 Mehrfachverwendung von Sichten . . . 10
3.2.2 Schubladendesign . . . 10
3.2.3 EAV Design . . . 11
3.2.4 Exemplarischer Aufbau des Dokuments (EAV) . . . 13
3.3 Zweite Iteration . . . 16
3.3.1 Administrationsstatus . . . 16
3.3.2 Erstellen von neuen Formularen . . . 17
3.3.3 Die Vorschau . . . 17
3.4 Dritte Iteration . . . 18
3.4.1 Administrationssystem . . . 18
3.4.2 Untersuchungen und Schließen eines Falls . . . 18
3.4.3 Bearbeitungsfunktion . . . 20
3.4.4 Nutzer Feedback . . . 20
3.5 Vierte Iteration . . . 22
3.5.1 Anpassung des Formular Erstellungssystems . . . 22
3.5.2 Refactoring . . . 22

4 Fazit und Ausblick **23**

Literaturverzeichnis **24**

A Anhang **25**

1 Einleitung

1.1 Motivation

Diese Bachelorarbeit beschreibt den iterativen Entwicklungsprozess einer Webanwendung für den Verein European Veterinary Society for Small Animal Reproduction (EVSSAR).
Bei genauer Betrachtung der Literatur zu seltenen Reproduktionskrankheiten von Tieren muss man zu der Erkenntnis kommen, dass die wissenschaftliche Datenlage nicht ausreicht. Nur selten sammelt sich innerhalb eines Instituts eine ausreichende Sammlung von Patientendaten.
Den Mitgliedern der EVSSAR werden aufgrund ihrer Spezialisierung häufig Tiere mit besonderen Erkrankungen vorgestellt. Ein Mittel, um diese Daten untereinander einheitlich zu erfassen und zu sammeln, fehlt jedoch. Ein Werkzeug, das den Austausch dieser Daten ermöglicht, ist der erste Schritt die Lage zu verbessern.
In der Tiermedizin gibt es erst seit einigen Jahren Initiativen für vergleichbare Anwendungen. In Abgrenzung zu dieser Arbeit verfolgen diese häufig andere Absichten. Ein Zweck ist das Sammeln der Motivation der Halter [5], oder Informationen über weit verbreitete Krankheiten und deren Regionalen Auswertung . [6]
Durch die Erhebung jeglicher Daten sind die Sammlungen für seltene Krankheiten schwer auswertbar. Die Konzepte, die hauptsächlich die Daten der Patientenverwaltungssoftware auswerten, sind nicht spezifisch genug und die eingegebenen Daten oft nicht eindeutig definiert. [1]

1.2 Ziel

Das Ziel dieser Arbeit ist die Entwicklung einer Webanwendung, die den Mitgliedern des EVSSAR ermöglicht, eine Fallsammlung für seltene Haustierkrankheiten zu erstellen. Dafür muss es möglich sein, verschiedene Formulare auszufüllen, Berichte zu editieren und Untersuchungen hinzuzufügen. Um dies zu gewährleisten, muss ebenfalls eine zugehörige Datenbank entworfen werden. Neben niedrigen Kosten soll die Seite möglichst unabhängig von Programmierern sein.

1.3 Kapitelübersicht

Das zweite Kapitel umfasst die Anforderungsanalyse und beschreibt Entscheidungen bei der Wahl der Werkzeugen und Sprachen.
In Kapitel drei wird auf den iterativen Prozess eingegangen. In jedem Unterkapitel werden die nächsten Funktionen, der Iteration, priorisiert. In zugehörigen Sektionen, werden Designentscheidungen genannt und aufgetretene Probleme besprochen.

Im vierten Kapitel ziehe ich mein Fazit und gebe Vorschläge zur Weiterentwicklung.

Der Code der Webanwendung kann über Github (https://github.com/robertb1984/BA.git/ Hash 3dfa73c9a77f886772597a149b3285ebb00bd090) geladen werden. Eine Anleitung zur Benutzung liegt bei.

2 Vorbereitungen

Bevor der Iterative Prozess beginnen konnte mussten einige Vorbereitungen getroffen werden. Um eine Entscheidung bei der Wahl der Technologien treffen zu können wurden die grundlegenden Anforderungen in einem Treffen definiert. In diesem Kapitel beschreibe ich das Resultat und gehe auf die gewählten Mittel ein.

2.1 Anforderungen

Um ein Produkt zu schaffen, das für die Zielgruppe nutzbar ist, entschied ich mich zur Aufnahme der Anforderungen für einen agilen Prozess. Als agilen Prozess wählte ich ein Verfahren das Feature Driven Development(FDD) ähnelt. [8] Meine Absicht war es, nach der Implementation grundlegender Funktionen, mich alle 14 Tage, mit meiner Bezugsperson zu treffen, um die Reihenfolge der nächsten Funktionen nach Priorität festzulegen.

2.1.1 Funktionale Anforderungen

Die Seite soll nur benutzbar für einen geschlossenen Kreis sein. Für jede Krankheit muss es ein eigenes Formular geben. Es soll möglich sein diese Formulare mit Daten zu füllen, sie zu speichern und ausgefüllte Dokumente zu lesen. Dem Ersteller muss es möglich sein, den Inhalt zu editieren. Außerdem soll es möglich sein Untersuchungen hinzuzufügen und die Fälle mit einem Abschlusskommentar zu schließen. Es soll Administratoren geben, die mehr Rechte und Funktionen haben. Diese Funktionen werden im weiteren Projektverlauf definiert.
Eine weitere Anforderung ist aufgrund des geringen Budgets, dass die Seite möglichst unabhängig von einem Entwickler verwaltet werden kann. Dies beinhalten die Funktion, dass Nutzer neue Formulare erstellen können.

2.1.2 Nichtfunktionale Anforderungen

Durch die geringe Mitgliederzahl und den dadurch geringen Andrang bestehen keine hohen Ansprüche an die Performance. Durch begrenzte finanzielle Mittel soll diese experimentelle Anwendung in ihrer Wartung und ihren Lizenzkosten so günstig wie möglich sein.
Ebenfalls erhielt ich zwei Beispieldokumente[1], die als visuelle Vorlage dienen. Durch diese strebte ich ein optisches Design an, das einem ausgedruckten Formular gleicht. Deswegen beschreibt im Verlauf dieser Arbeit der Ausdruck „Formular “ die Vorlage für einen Krankheitsfall. Das ausgefüllte Formular, das einzelne Dokument oder der Report/Bericht dient als Bezeichnung für den gespeicherten Fall.

[1] Im Anhang

2.2 Auswahl der Sprache

Zur Auswahl der Sprache dienten folgende Rahmenbedingungen: :
1. Ein zugehöriger Server soll möglichst kostengünstig sein
2. Sie soll ebenfalls verbreitet sein, damit sich Programmierer finden lassen, die die Seite weiterentwickeln
3. Die Performance ist relativ unwichtig
4. Die Möglichkeit der Anbindung eines kostengünstigen Datenbankverwaltungssystems soll bestehen

Aufgrund der oben genannten Punkte entschied ich mich nach einigen Nachforschungen für PHP. PHP ist immer noch eine Einstiegssprache, für erste Erfahrungen der Website Programmierung. Durch das erstarken vieler Frameworks wie Laravel ist sie auch verbreitet [10] und damit das finden eines Nachfolgers leicht. Eine PHP Webanwendung läuft auf jedem Linux system und erlaubt die Anbindung nahezu jedes Datenbankverwaltungssystems, welches für diese Arbeit in Frage kam.

2.3 Auswahl des Datenbankverwaltungssystems und der Engine

Als Datenbankverwaltungssystem hatte ich bis jetzt lediglich Erfahrungen mit MySQL (GPL).
Um Lizenzprobleme zu vermeiden wählte ich MariaDB (LGPL) als Verwaltungssystem. So konnte ich die Vorteile der Weiterentwicklung von MySQL nutzen. [2]Um Transaktionen benutzen zu können und weil die Geschwindigkeit zu vernachlässigen war, wählte ich InnoDB als Engine.

2.4 Auswahl des Frameworks

Die Wahl eines Frameworks erschien naheliegend. Es erspart Zeit, da der Programmierer auf viele Grundfunktionen zugreifen kann und diese nicht selbst implementieren muss. Gleichzeitig bietet es viele getestete Sicherheitsfunktionen. Bei der Wahl des Frameworks ist die Einfachheit und geringe Einarbeitungszeit für komplexere Datenbankanfragen, entscheidend. Schließlich versprachen die Anforderungen, dass die Anwendung hauptsächlich CRUD Aufgaben erledigt. Eine lange Einarbeitungszeit oder zögerliche Wahlentscheidung würde den Geschwindigkeitsvorteil negieren.

2.4.1 CakePHP

Ich beschäftigte mich zuerst mit CakePHP. Im Grunddesign sind die Anfänge der Datenbank deswegen an die Cake Namenskonvention angepasst. Während es einfach war eine Tabelle zu erstellen und eine passende Seite für diese zu „baken“

[2]https://www.gnu.org/licenses/gpl-3.0.de.html

[3], waren komplexere Datenbankanfragen mithilfe ihrer Dokumentation nicht einfach umzusetzen. Durch ihre eigenen Funktionen für komplexere Anfragen muss ein Einsteiger diese Semantik neu erlernen.

2.4.2 Codeigniter

Das nächste Framework war Codeigniter 3. Ein sehr schlankes Framework, das sich nur mit dem Nötigsten installiert. [4] Ohne eine automatische Codegenerierung gestalten sich die ersten Schritte schwieriger, dafür sind diese schnell nachvollziehbar. Die Dokumentation war ebenfalls deutlich besser als bei CakePHP und die Einstiegshürde niedriger.

2.4.3 Laravel

Das nächste Framework war Laravel.
Laravel ist vergleichbar mit CakePHP und bietet gleiche Herausforderungen. Ich konnte in ihrer Dokumentation keine Möglichkeit finden eine eigene Datenbankabfrage zu nutzen. Die Dokumentation im allgemeinen ist vergleichbar mit der von Codeigniter. Ein Selbstest, bei dem es die Aufgabe war, zwei Tabellen zu verbinden und das Resultat auf einer neuen Seite auszugeben, war nicht unproblematisch und benötigte trotz guter Dokumentation viel Zeit.
Ich entschied mich Aufgrund der aufgezeigten Vorteile für Codeigniter. Wenn es um den Kontakt mit dem ersten MVC Framework geht, sind andere zu gleichem Schluss gekommen.[3]

2.5 Erstes Datenbankdesign

Um die ersten Frameworks zu testen, wurde das Grunddesign der Datenbank erstellt.[5] Die größte Frage stellte sich bei dem System für die Datenaufnahme der spezifischen Krankheiten. Betrachtet man die Beispieldokumente[6], erkennt man, dass es immer einen Teilblock für das Tier und einen für die Behandlung gibt. Der Unterschied liegt in beiden Dokumenten bei den nötigen Daten für die jeweilige Krankheit. Das größte Problem würde es sein, diese unterschiedlichen Datensätze für jeden Krankheitsblock zu handhaben. Eine Lösung war eine Tabelle für jede Krankheit, in der die Resultate gespeichert werden.
Das erste Design ist in Abbildung 1 dargestellt.:

[3] Das Thema Backen wird gnadenlos durchgezogen.
[4] Selbst composer ist nicht zwingend nötig.
[5] Ich erstellte dieses parallel zu dem CakePHP Test.
[6] Im Anhang

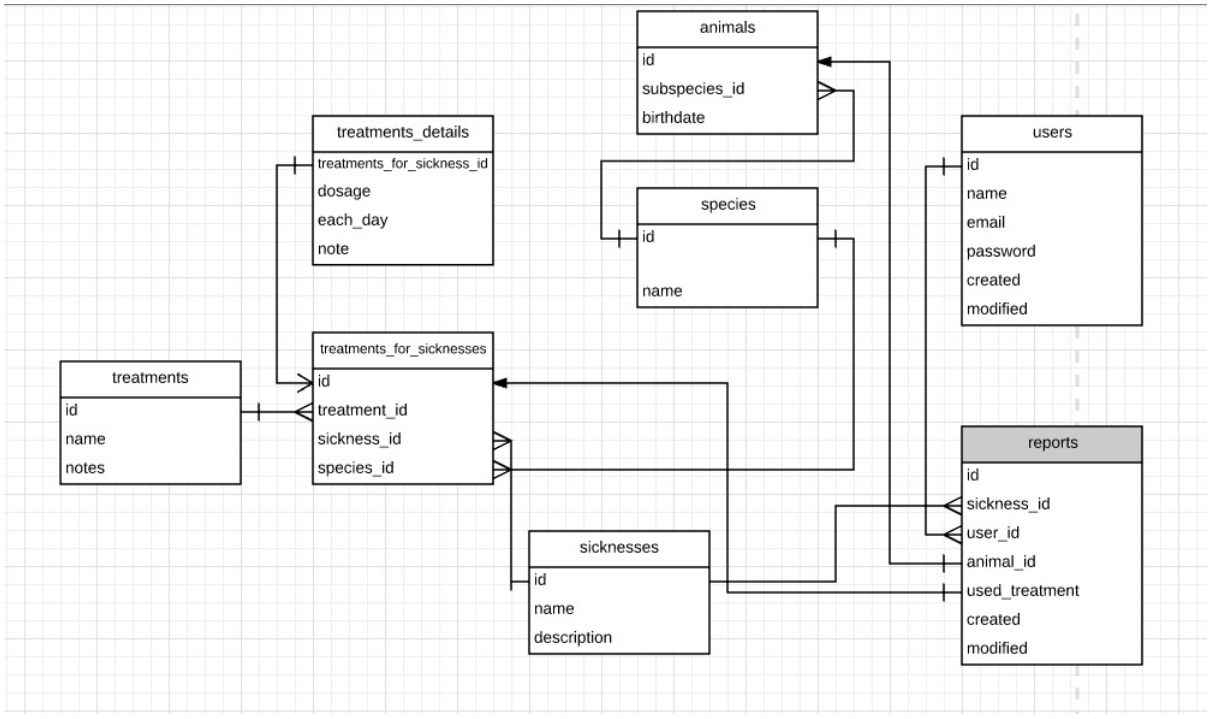

Abbildung 1: Das erste Design der Datenbank erstellt in Lucidcharts

.

„treatments“ : Eine Tabelle für die verschiedenen Behandlungsmethoden, z.B. Medizin oder eine einfache Massage.

„treatments_for_sicknesses“ : Eine Beziehungstabelle, in der die Behandlung gespeichert wird. Dies gibt dem Nutzer die Möglichkeit, die gleiche Behandlung erneut zu wählen und vermeidet das erneute ausfüllen dieses Dokumententeils.

„treatments_details“ : Hier werden Details wie Dosierung erfasst . Diese Sollte es nicht für jede Behandlung geben und konnte deswegen in eine zusätzliche Tabelle ausgelagert werden.

„sicknesses“ : Eine Tabelle, die die Krankheit beschreibt.

„reports“ : Der Fall mit all seinen Beziehungen zu anderen Tabellen.Es handelt sich um eine Fremdschlüsselsammlung, die diesen Fall definiert. Da die genaue Handhabung zu diesem Zeitpunkt noch unklar ist, fehlen angebundene Tabellen für den Krankheitsblock.

„animals “: speichert das Tier für einen Fall.

„species“ : Speichert die Spezies wie z.B. der Hund.

„users“ : Tabelle, um die Nutzer zu verwalten.

3 Ausführung

Zu Beginn jeder Iteration gehörte ein Treffen mit meiner Bezugsperson Dr. Sebatian Artl. In einer Demonstration wurde der Status des Projekts vermittelt. Anhand diesem bestimmten wir die Anforderungen der Funktionen für die nächste Iteration. Dieses Kapitel beschreibt die Iterative Entwicklung. Jedes Unterkapitel geht auf unsere Bestimmung der Anforderungen, der geplanten Funktionen für die jeweilige Iteration, ein. Anschließend dienen Unterkapitel innerhalb einer Iteration dazu, technische Entscheidungen zu erläutern.

3.1 Grundstein

Die Grundlage für die weiteren Iterationen sollte eine Ausgabe der ersten Tabellen der Datenbank, ein Loginsystem und die erste Version der Datenbank umfassen. Um die Anwendung visuell ansprechender darzustellen, wählte ich ein css für Bootstrap von https://bootswatch.com/ [7]. Ich probierte einige aus und behielt Flatly und Cerulean.

Als nächstes implementierte ich das Loginsystem nach dem Vorbild eines Tutorials [8]. An einigen Stellen nahm ich dabei Veränderungen vor. MD5 erschien mir nicht sinnvoll. Stattdessen benutzte ich Bcrypt. Dieser Algorithmus bietet Sicherheiten wie Salt und ist bei der Verschlüsselung vergleichsweise langsam, was die Seite sicherer gegen Brute-Force-Attacken macht.[4] Bei einem erfolgreichen Login werden die Benutzerdaten in die Sessionvariablen eingeladen. Um in Zukunft in jedem Controller überprüfen zu können, ob der Benutzer authentifiziert wurde, erweiterte ich den CI_controller, von dem jeder Controller erbt. Dieser erhielt die Funktion 'is_authenticated()', die ich nun in jedem Konstruktor verwenden konnte oder zum Sperren von Funktionen.

Nach dem Vorbild für das Cryptorchidism Formular erstellte ich eine einzelne Sicht, die der Vorlage entsprach. Einzig den Behandlungsblock (Treatment) veränderte ich von einer festen Anzahl von Markierungsfeldern zu einem Aufklappmenü mit Eingabefeldern für die Dosierung, den Zeitraum, und die Anzahl der Einnahmen pro Zyklus, den man auch definieren konnte. Dieses System ähnelte den Einnahmeanweisungen für Menschen zu stark.
Eine beschränkte Anzahl von Markierungsfeldern war jedoch ungünstig, da die Seite, unter anderem, ermöglichen sollte neue Behandlungsmethoden zu finden. Das Datenbankdesign erlaubte mir, neben neuen Behandlungen, alte vorzuschlagen, die für die gleiche Krankheit schon einmal benutzt wurden. Ebenfalls sollten neben der Beschreibung des Mittels die Werte der erstellten Behandlung eingeladen werden. Hierzu beschäftigte ich mich mit Javascript und der PHP Kommunikation. Nachdem ich über Ajax dieses Problem lösen konnte, passte ich ebenfalls die Krankheitssuche an, damit diese die Beschreibung der Krankheit ausgibt.
Abschließend implementierte ich die Speicherung der Daten und eine Übersicht

[7] Die gewählten sind unter MIT veröffentlicht.
[8] https://imron02.wordpress.com/2013/06/01/simple-login-using-codeigniter-database/ (Stand 29.07.2017)

über die eigenen erstellten Berichte.

3.2 Erste Iteration

Ich konnte Dr. Sebastian Arlt bei unserer ersten Vorbesprechung eine relativ funktionale Anwendung präsentieren, mit mehr Funktionen als wir eigentlich gehofft hatten. So war es möglich ein neues Dokument zu erstellen und die Eingaben zu speichern. Noch ausstehend war eine Möglichkeit diesen Bericht einzuladen und auf der Seite zu sehen.
Bei dem ersten Treffen für unsere Iterationen wurden einige Punkte angesprochen: 1. Die Frage nach einem Administrationssystem. 2. Der Grund für die Umstrukturierung der Behandlungen war nachvollziehbar. Jedoch wies er mich darauf hin, dass in der Tiermedizin die Angaben der Dosierung pro Einnahme in Kombination mit der gesamt Tagesanzahl und der Anzahl der Einnahmen ausreichend ist. 3. Eine Funktion um nach Berichten zu suchen, muss erstellt werden. In diesem Zusammenhang soll es eine Möglichkeit geben, sich ausgefüllte Berichte anzusehen. Wir beschlossen die folgende Priorisierung der Funktionen. 1. Das Auslesen von ausgefüllten Dokumenten. 2. Eine Möglichkeit nach Berichten zu suchen. 3. Die Planung der Umsetzung weiterer Formulare. 4. Idee für ein geeignetes Administrationssystem. 5. Die Überarbeitung des Behandlungsblocks. Die Tage nach unserem Gespräch erledigte ich schnell die ersten zwei Aufgaben.

3.2.1 Mehrfachverwendung von Sichten

Um die gleiche Sicht sowohl für das Erstellen als auch das Auslesen zu benutzen und so bei einer möglichen Änderung weniger Sichten anpassen zu müssen, benutzte ich für jede Eingabe ähnliche Strukturen. Diese Struktur ist in Listing 1 dargestellt.

Listing 1: PHP Code

```
echo form_input('name', set_value('name',isset($data['name']) ?
    $data['name'] : ''), $readonly);
```

Es wird überprüft, ob eine Variable übergeben wurde und falls dies der Fall ist, wird diese angezeigt. Ansonsten ist die Eingabe leer. Wenn der Seitenaufruf lediglich dem auslesen dienen soll, wird das Eingabefeld auf „nur lesen “ gesetzt. Gleichzeitig wird im Controller beim einladen der Sicht eine variable gesetzt, die alle Eingaben blockiert, falls es sich um ein ausgefülltes Formular handelt.

3.2.2 Schubladendesign

Die größere Aufgabe war es Nutzern das Design zukünftiger Formulare zu ermöglichen. Ich hatte bereits erfolgreich getestet, dass ich die Funktionalität für ein System mit strikt entworfenen Tabellen umsetzen konnte.
Nun ging es darum eine Möglichkeit zu finden die es den Nutzern ermöglicht neue Formulare zu entwerfen. Durch die Auseinandersetzung hatte ich jedes Grundformular in drei Teile eingeteilt (siehe Abschnitt 2.5). Die Formulare für die unterschiedlichen Krankheiten unterscheiden sich durch den Krankheitsblock. Ein Aufteilen in drei unterschiedliche Sichten war der erste logische Schritt, um mich von der Handhabung als ganzes Dokument zu entfernen und so die Logik in PHP

der Datenbank anzunähern.

Listing 2: PHP Code

```
$this->load->view('templates/header');
$this->load->view('reports/cryptorchidism',$data);
$this->load->view('templates/footer');
```

Der Aufruf der Sichten wurde folgendermaßen angepasst:

Listing 3: PHP Code

```
//$sichness = der name der jeweiligen Krankheit
$this->load->view('templates/header');
$this->load->view('reports/animal',$animal_init);
$this->load->view('reports/'.$sickness,$form_data);
$this->load->view('reports/treatment',$data);
$this->load->view('templates/footer');
```

Meine Absicht war es einerseits auf Veränderungen der Blöcke besser reagieren zu können. Falls es in den nächsten Iterationen Veränderungen an einem der statischen Blöcke geben sollte, wäre jetzt nur die Anpassung einer Sicht nötig. Andererseits kann anhand des Namens der Krankheit eine passende Sicht für jede Krankheit benutzt werden. Dafür muss es lediglich zugehörige Tabellen geben, die den Namen der jeweiligen Krankheit besitzen.
Die Idee sah vor, mithilfe einer Sicht den Aufbau des Krankheitsblocks zu definieren und für diesen eine Tabelle mithilfe von Codeigniters Database Forge [7] zu erstellen. Diese Tabelle muss nach dem „name “ Eintrag der „sickness “ Tabelle benannt sein. Mir wurde jedoch schnell deutlich, dass dieser Ansatz eine große Lücke hatte. Diese individuelle Sicht für jeden Krankheitsblock müsste trotzdem noch implementiert werden. Gleichzeitig müssten Werte für die Validierung ebenfalls aus der Datenbank geladen werden und nicht Fest im Code stehen.
Diese Lösung war also folglich nicht was ich mir erhofft hatte. Die Trennung der Blöcke erwies sich jedoch als gut. Es ermöglichte mir die Auslagerung von Vorbereitungsfunktionen in der Refaktorisierung (siehe Abschnitt 3.5.2) und gestaltete so den Code übersichtlicher.

3.2.3 EAV Design

Nachdem der erste Ansatz nicht zum gewünschten Ergebnis geführt hatte, musste eine weitere Lösung gefunden werden. Man kann alles nötige in zwei Attributslisten speichern. Die Erste umfasst alle Definitionen. Dazu gehören sowohl die Validierungsregeln als auch der Name, nachdem ich bei der Übergabe suchen konnte und die Frage/Beschreibung, die vor dem Eingabefeld im Formular stehen würde. Die Attributsammlung für eine Krankheit beschreibt die Spalten der „imaginären“ Tabelle und die Validierungsregeln des Attributs. Die Zweite Liste beinhaltet die gespeicherten Eingaben für jedes Attribut für jeden Bericht.
Zur Lösung zog ich das Entity-Attribute-Value Modell (EAV) heran . Eine Datenbank im kompletten EAV Design war für unsere Verwendung unnötig. Nur für die Attributsliste der Definition, also für die Tabellen die festlegen wie der

Krankheitsblock aussehen würde, wäre das EAV Modell nutzbar. Die restliche Datenbank wäre immer noch in Normalform.

Valentin Dinu und Prakash Nadkarni(2006) [2] schlussfolgerten:
„As stated earlier, the use of EAV modeling is appropriate for the following situations:
-Data are sparse, heterogeneous, have numerous attributes and new attributes are often needed. This is seen for databases with a clinical data repository component, especially those spanning multiple clinical specialties.
-The number of classes is very large, and numerous classes have a very modest number of instances, even if the attributes for these classes are not necessarily sparse. This is seen in ontology/taxonomy systems and certain bioscience schemas (e.g., neuroscience).
-There are numerous hybrid classes, possessing both sparse and non-sparse attributes. Typically, not all classes in the data will meet the requirements for EAV modeling.“ (Kapitel 6)
Alle drei Punkte erfüllt unser Problem.

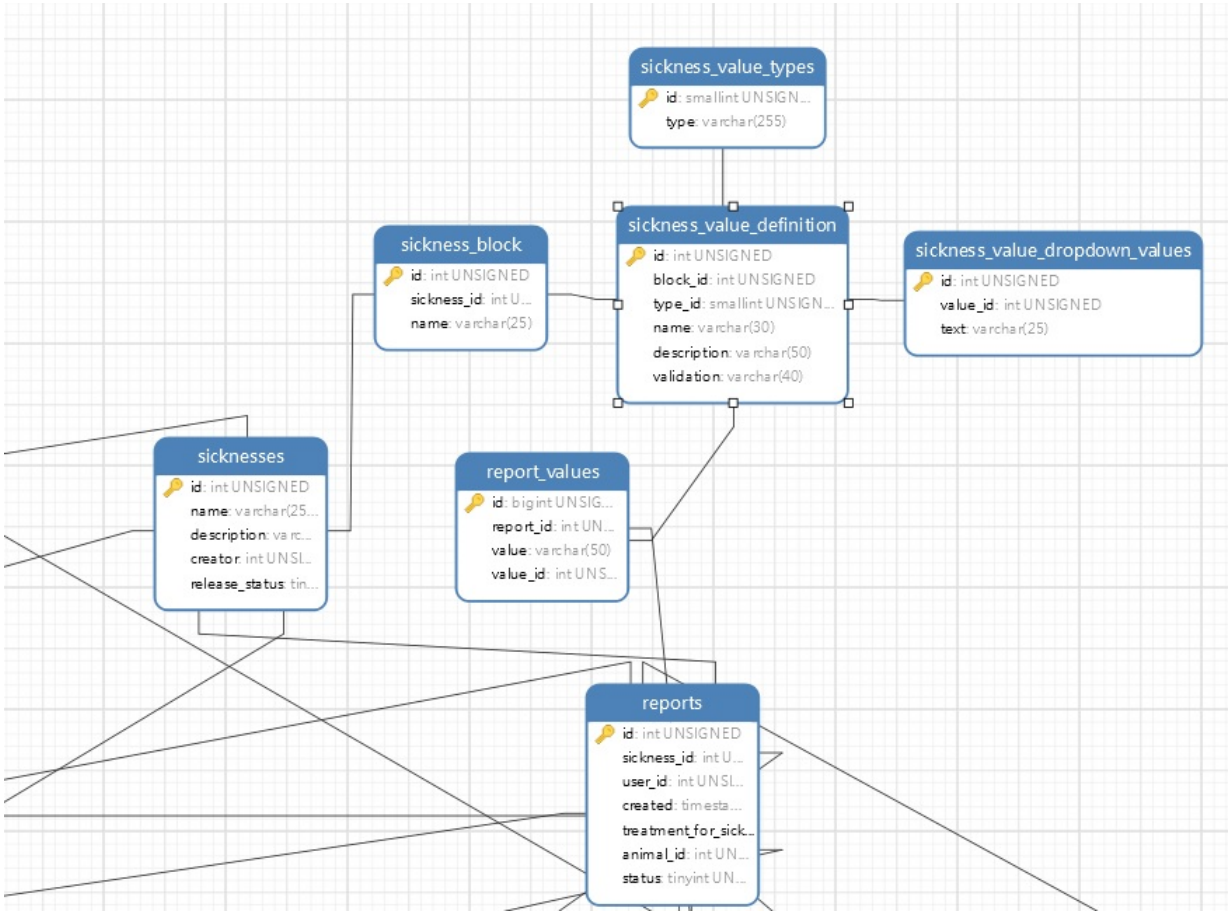

Abbildung 2: EAV Krankheitsblock erstellt mit Navicat Pro

Abbildung 2 zeigt den nun neuen EAV Datenbankausschnitt :
„sicknesses“ und „reports“ sind zum ersten Datenbankdesign gleich geblieben. Reports dient als Fremdschlüsselsammlung mit den Referenzen für die ausgefüllten Dokumente. Und sicknesses beinhaltet die Grundwerte für ein Formular.
„sickness_block“ beinhaltet die Unterblöcke des Krankheitsblocks, wie im Beispielformular für den linken und rechten Hoden[9].

„sickness_value_definition“ beschreibt die Definition von jedem Eingabefeld des zugehörigen Unterblocks also das Attribut.

„sickness_value_types“ wurde hauptsächlich für die spätere Auslagerung der Validierungsregeln erstellt.

„sickness_value_dropdown_values“ eine Tabelle die alle Werte für den speziellen Typ Aufklappmenü bereitstellt.

„report_values“ dient zur Speicherung der Einträge für das jeweilige ausgefüllte Dokument, also die „values“ .

Anders als in den gängigen EAV Beispielen [10] haben wir hier keine Benutzer die beliebig unterschiedliche Attribute erhalten können. Die „sickness“-Tabelle mit ihren zugehörigen EAV Tabellen dient eher als Objektdefinition, um keine neuen Tabellen anlegen zu müssen. Die erstellten Objekte lassen sich wiederum in „reports“ und „report_values“ finden.

3.2.4 Exemplarischer Aufbau des Dokuments (EAV)

Die Daten zu den Einträgen werden durch die Modellfunktion load_input_fields geladen.

Listing 4: PHP Code

```
//Report_model Zeile 349
 function load_input_fields($id)
        {
            $query = $this->db->query('
                SELECT sickness_value_definition.id ,type,
                   sickness_value_definition.name ,
                   sickness_value_definition.description,
                   validation, sickness_block.name AS blockname
                FROM sickness_value_definition
                JOIN sickness_block ON sickness_block.id =
                   block_id
                LEFT JOIN sickness_value_types ON
                   sickness_value_types.id =
                   sickness_value_definition.type_id
                WHERE sickness_id = '.$id.'
                 ;');
            return $query->result_array();
```

[9] Cryptorchidism Formular im Anhang
[10] https://www.youtube.com/watch?v=X5Q9G3pa7I4 (Stand 26.07.2017)

```
        }
```

Nachdem das Array mit den Daten gefüllt wurde, werden im Controller mit diesem Teil des Codes alle benötigten Arrays für die Aufklappmenüs gefüllt.

Listing 5: PHP Code

```
$form_data['entries'] = $this->Report_model->load_input_fields(
   $sicknessID);
  foreach ($form_data['entries'] as $value) {
    if($value['type'] == "drop")
          {
              $form_data[$value['name']]= $this->Report_model-
                   >get_dropdown_data($value['id']);
                $form_data['dropdowns'][$value['name']]= $this
                   ->dropdownData($form_data[$value['name']], '
                   text', $value['description']);
            }
    }
```

In der Sicht werden die Daten mithilfe dieser „foreach“ Schleife aufgebaut.

Listing 6: PHP Code

```
 foreach ($entries as $entrie)
    {
        if ($entrie['blockname'] != $group)
        {
            $group = $entrie['blockname'];
            echo form_label($group." : ",'',$attributesGroup );
            echo "<br/>";
        }
        switch ($entrie['type'])
        {
            case 'text':
                echo form_label($entrie['description'] ,
                   $entrie['name'] );
                echo form_input($entrie['name'] , set_value(
                   $entrie['name'],isset($entrie['value']) ?
                   $entrie['value'] : ''),$readonly);
                echo "<br/>";
                break;
            case 'drop':
                echo form_dropdown($entrie['name'],$dropdowns[
                   $entrie['name']], set_value($entrie['name'],
                   isset($entrie['value']) ? $entrie['value'] :
                    ''),$disabled);
                echo "<br/>";
                break;
        }
    }
```

Der Fall „text“ fängt sowohl Zahleneingaben als auch Texteingaben ab, denn diese unterscheiden sich lediglich in den Validierungsregeln. In der Speicherungsfunktion werden die Informationen zu der passenden Krankheit aus der Datenbank geholt. Aus dieser erhält das Skript die Namen der abgeschickten Variablen (Post). Die zu speichernden Daten werden über ihren Namen verbunden und mit den eingeladenen Validierungsregeln geprüft.

Listing 7: PHP Code

```
$sicknessID = $this->input->post('sickness');
            $entries = $this->Report_model->load_input_fields(
                $sicknessID);
            foreach ($entries as $entrie)
            {
                $this->form_validation->set_rules($entrie['name
                    '],$entrie['name'],$entrie['validation']);
            }
```

3.3 Zweite Iteration

Am Anfang der zweiten Iteration konnte ich eine Arbeit vorweisen, die aus Benutzersicht auf dem gleichen Stand war, wie bei der ersten. Durch einen Fehler[11] konnte man für alte Formularvorlagen immer noch nicht die gespeicherten Eingaben anzeigen. Das neue EAV System war auf dem gleichen Stand wie das Alte. Ich konnte jedoch die Vorteile erklären und warum die Zeitinvestition wichtig war. Nach meiner Erklärung waren wir uns einig, dass das Design nur Sinn machen würde, wenn wir eine geeignete Seite zur Erstellung implementieren können. Speziell die Frage, ob wir am Ende ein Werkzeug haben, mit dem die Formularerstellung verständlich ist, erschien uns beiden fraglich. Da ohne diese Möglichkeit mein Ziel der Autonomie der Seite unmöglich war, wurde dies auch die priorisierte Aufgabe, für die nächste Iteration.
Darüber hinaus haben wir ein Administrationssystem definiert. Der normale User sollte nur die Möglichkeit haben, beliebige, definierte Formulare auszufüllen und nach ausgefüllten Dokumenten anderer Nutzer zu suchen. Ein Administrator sollte alle anderen Aufgaben übernehmen. Nur der Administrator sollte diesen Status vergeben können. Ich habe auf das Risiko aufmerksam gemacht, das ein einzelner Administrator enormen Schaden anrichten könnte und diesen nur durch eine Person zu ernennen, erschien unsicher. Meine ursprüngliche Absicht war es, dass für die Erhebung mindestens zwei Stimmen nötig wären. Es wurde jedoch beschlossen das bei der Anzahl der Mitglieder und der erwarteten niedrigen Nutzung diese Form, für die Anfangsphase, ausreichend ist.
Wir einigten uns so auf folgende Prioritätsliste:
1. Administratoren ermöglichen neue Formulare zu erstellen.
2. Ein funktionierendes Auslesen ausgefüllter Dokumente.
3. Eine Administrationslogik implementieren.
4. Die Überarbeitung des Behandlungsblocks.

3.3.1 Administrationsstatus

Für den Administrationsstatus fügte ich in die Benutzertabelle einen Bitwert hinzu. Die Zugriffskontrolle funktionierte Äquivalent zu dem Login. Ich erweiterte ebenfalls den CI_Controller um eine Funktion, die die geladenen Session-variablen auf den Administrationsstatus überprüft. Diese Funktion konnte nun entweder im Konstruktor eines Controllers, um alle Funktionen des Controllers zu sperren oder am Anfang einer Funktion aufgerufen werden.

[11] Ich hatte vergessen eine neue Datei auf mein Notebook zu ziehen

3.3.2 Erstellen von neuen Formularen

Um neue Formulare erstellen zu können, musste ich mir zuerst Gedanken über den Prozess machen. Relativ angenehm war, dass ich davon ausgehen konnte, dass nur Administratoren Zugriff haben sollten. Folglich konnte dieser Prozess etwas komplizierter sein, schließlich sollten diese Administratoren eingewiesen sein. Trotzdem musste ich einen Kompromiss aus Zugänglichkeit und Umsetzbarkeit finden, da ansonsten die Bereitschaft sinken würde neue Formulare zu erstellen. Die erste Idee war eine Vordefinition. Der Benutzer definiert die Blockanzahl und die Anzahl der Werte pro Block. Auf der nächsten Seite baut sich eine Tabelle auf und der Benutzer kann Werte, mit den Definitionen für die einzelnen Zeilen füllen. Dieses System ist schnell umsetzbar, jedoch zu statisch und für den Nutzer nicht intuitiv.
Ich entschied ich mich dafür eine Seite zu implementieren, die es ermöglichen sollte, ein Formular von oben nach unten, zu definieren. Nach Abschluss durch den Nutzer kommt eine Vorschau, denn bei einem Fehler ist dieses neue Formular direkt im System.
Für die Lösung orientierte ich mich an einem Beispiel von Marco Folio[12]. Dieses erweiterte ich um einige Funktionen.
Im Resultat muss der Benutzer zuerst einen Block definieren. Nachdem ein offener Block existiert, kann er die Werte bestimmen, nach denen gefragt werden soll. Dafür muss er beim hinzufügen der Zeilen einen Variablennamen angeben und kann dann die Frage und den Typ definieren. Der Eingabetyp „number“ unterscheidet sich nur in der Validierungsregel von einer Texteingabe. Lediglich für die Aufklappmenüs musste ein weiterer Knopf implementiert werden. Die Werte des Aufklappmenüs muss der Benutzer in einen Text schreiben und sie durch Kommata trennen, damit diese getrennt werden können.

3.3.3 Die Vorschau

Zur Erzeugung einer genauen Vorschau wollte ich die gleichen Sichten wie bei der gewöhnlichen Formularausgabe benutzen. Ansonsten kann das echte Resultat abweichen. Mit der Bestätigung des Benutzers durch einen Klick auf den Vorschauknopf wird die gesetzte Definition gesendet. Aus den Daten wird im Form Controller ein Array gebaut, das äquivalent zu den eingeladenen Definitionen bei einem gewöhnlichen Aufruf eines leeren Dokuments ist.

[12] http://www.marcofolio.net/webdesign/expanding_a_html_form_using_javascript.html (Stand 20.07.2016)

3.4 Dritte Iteration

Zu Beginn der dritten Iteration konnte ich zeigen das die autonome Webanwendung möglich ist. Der Kreislauf von Formularerstellung bis Fallausgabe war durchführbar und das ohne einen Programmierer. Noch offen war neben der gesamten Logik für Untersuchungen und dem Abschluss eines Falls, die Einrichtung eines Servers und die Bearbeitungsfunktion. Für den vollen Funktionsumfang musste es noch die Möglichkeit geben Fälle zu schließen. Beim Schließen wäre es wünschenswert, wenn der Ersteller einen Abschlusskommentar hinzufügen könnte.

Für die Untersuchungen hatte ich bereits ausführliche Anforderungen erstellt. In den Beispieldokumenten, fragten diese nach den gleichen Werten wie der Krankheitsblock. Dies stellte sich nur als Zufall heraus. In unseren Gesprächen stellten wir fest, dass dies nicht der Fall sein musste. Die Untersuchungen könnten auch keine Ähnlichkeiten mit dem Krankheitsblock haben, oder nur nach Teilen des Krankheitsblocks fragen.
Die Bearbeitungsfunktion war bisher auch nicht relevant, da diese erst sinnvoll ist, wenn man ganze Dokumente ausfüllen kann. Zuvor fehlten immer noch Untersuchungen und der Abschluss.
Für die Beschränkung des Zugriffs war mein Vorschlag, die Erstellung eines neuen Accounts nur Benutzern zu ermöglichen, deren E-Mail in einer Liste in der Datenbank steht. Diese Liste müssen Administratoren bei Bedarf erweitern.
Wir beschlossen folgende Priorisierung:
1. Die Untersuchungen implementieren
2. Das schließen eines Falls
3. Die Bearbeitungsfunktion
4. Das Administrationssystem

Das einrichten eines Servers gestaltet sich mit Codeigniter als ziemlich einfach. In einigen wenigen Schritten kann man die Seite auf einem Server einsetzen.[13]

3.4.1 Administrationssystem

Bis jetzt waren alle Administrationsfunktionen in einer weiteren Zeile in der Navigationsleiste enthalten. Dies war optisch nicht unbedingt ansprechend und wegen der vielen weiteren Administrationsfunktionen auch nicht mehr tragbar. All diese Funktionen wurden in ein Aufklappmenü verlagert. Zu der Umsetzung benutze ich ein ausführliches Tutorial[14].

3.4.2 Untersuchungen und Schließen eines Falls

Anhand der ermittelten Anforderungen mussten eigene EAV Tabellen für die Untersuchungen erstellt werden. Das Datenbankdesign sieht genauso aus wie das für den Krankheitsblock, mit einer zusätzlichen Tabelle im Resultat zwischen der report Tabelle und den Resultaten der Untersuchungen. Diese Tabelle bietet die Möglichkeit zwischen den Untersuchungen zu unterscheiden.

[13] Anleitung in der Git Ablage oder in der Dokumentation für Codeigniter.
[14] https://www.youtube.com/watch?v=m_xnqEgHnWM (Stand 15.07.2017)

Die Tabellen wurden ebenfalls nach dem EAV-Modell entworfen. Das Werkzeug zum Erstellen sollte sich jedoch von dem der Krankheitsblöcke unterscheiden. Um den Prozess zu vereinfachen, wollte ich dem Ersteller ebenfalls eine einfachere Möglichkeiten bieten. Auf einer neuen Seite muss man sich eines der Formulare aussuchen, das noch keine definierten Untersuchungen hat. Nach einer Weiterleitung hat der Benutzer drei Optionen. Er kann, wie bei dem Krankheitsblock, jeden Eintrag ausführlich definieren, eine Untersuchung genauso aussehen lassen wie den Krankheitsblock oder sich explizit Unterblöcke des Krankheitsblocks aussuchen, die in den Untersuchungen gefragt werden.

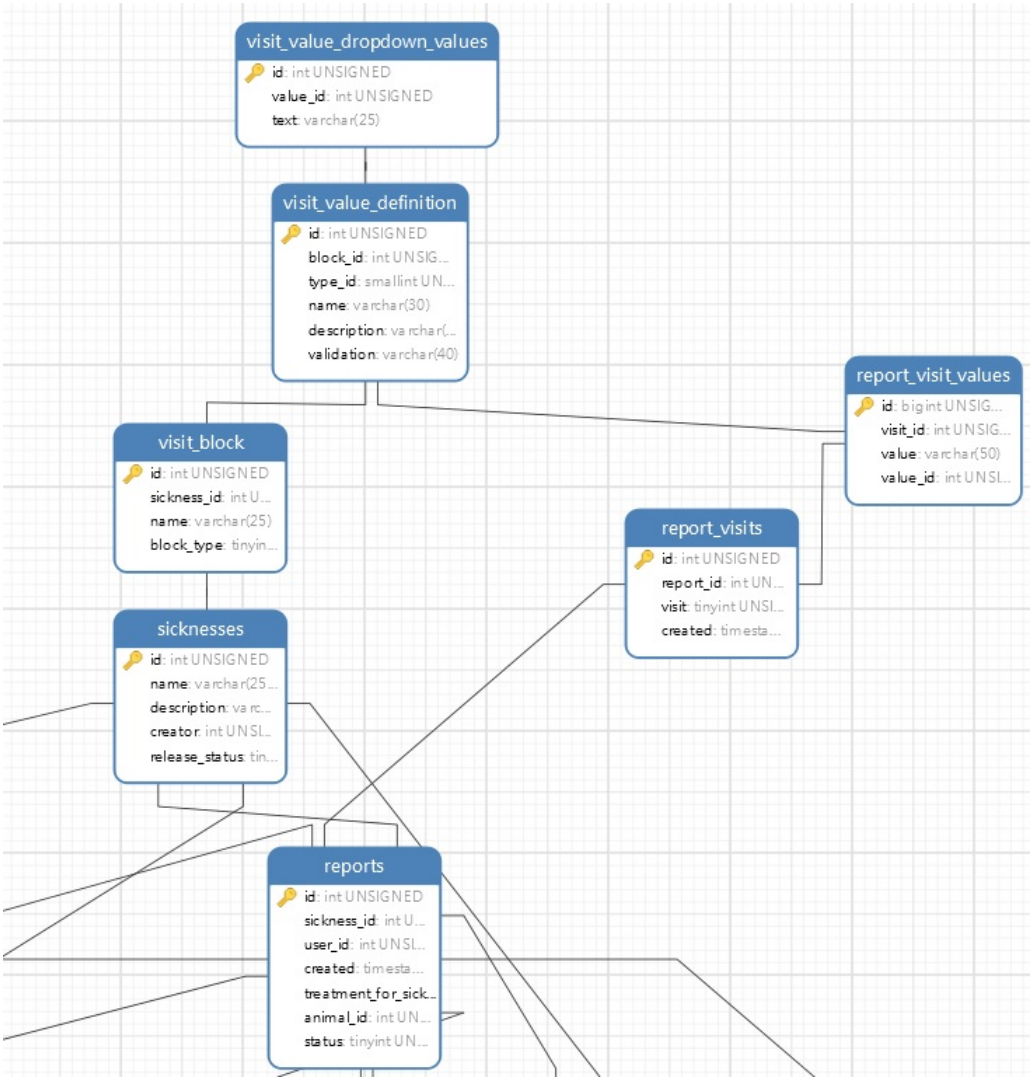

Abbildung 3: Datenbankstruktur für Untersuchungen erstellt in Navicat Pro

.

Erneut dienen als Ausgangspunkt sicknesses und reports.

„report_visits “ Im Vergleich zum Design des Krankheitsblocks ist diese Tabelle

der entscheidende Unterschied. Es handelt sich um eine eingefügte Tabelle, die die Ergebnisse dem jeweiligen Besuch zuweist.
„visit_block “ dient als Verbindung zwischen den Blöcken und den zugehörigen Werten.

„visit_value_definition “ umfasst die Definitionen der einzelnen Zeilen.

„visit_value_dropdown_values “ beinhaltet die Werte für jedes Aufklappmenü.

„report_visit_values “ erfasst die eingetragenen Werte für jede Untersuchung von jedem Fall.

Aus Zeitgründen wurde aus dem Schließen eines Falls in der Datenbank an den „reports“ eine weitere Tabelle erstellt und im Formular ein Abschlusskommentar mit einer Abschlusseinschätzung erfragt. Hier könnte ebenfalls eine EAV Tabelle benötigt werden (siehe 4).

3.4.3 Bearbeitungsfunktion

Die Umsetzung dieser Funktion ist sehr komplex und kann daher viele Probleme auslösen. Das Formular konnte nun aus allen drei Grundblöcken bestehen, einem Abschlusskommentar und zusätzlich einer unbekannten Anzahl von Untersuchungen die in der Theorie alle die gleichen Variablennamen hätten.
Ich verwarf aufgrund der Komplexität die Möglichkeit nur veränderte Einträge zu speichern. Die Implementierung aktualisiert alle Einträge, egal ob diese verändert wurden oder nicht. Dies war die sicherste Lösung. Bei der Speicherung von Untersuchungen ist das größte Problem das jede Untersuchung die gleichen Variablen besitzt und somit, die Namen bei der Post Übergabe nicht eindeutig sind. Dies habe ich behoben, indem ich an jeden Namen der Variablen die Kennzahl der Untersuchung anhing.

Ich nutzte die gleiche Funktion für die Erweiterung der Untersuchungen für die neue Bearbeitungsfunktion und für das Schließen. Eine Funktion mit über 100 Zeilen, die verschiedene Fälle behandelt, ist unleserlich und macht eine Weiterentwicklung schwierig. Deswegen sollte sie in der 4. Iteration überarbeitet werden.

3.4.4 Nutzer Feedback

Um Benutzerresonanz zu erhalten, setzte ich einige vor die Aufgabe die Seite zu benutzen. Von den sechs Testpersonen erklärte ich vier Personen die genaue Funktionsweise jeder Seite und Zweien nur den Verwendungszweck. Die Eingewiesenen konnten die Seite ohne größere Probleme benutzen. Es konnten nützlicherweise einige kleinere Fehler gefunden werden. Dieser Test spiegelte gut die erwartete Realität, denn Administratoren sollten ebenfalls eingewiesen sein und wissen wie die Anwendung zu benutzen ist. Wichtiger und interessanter waren die Tests ohne Erklärung. Durch diese konnten einige kritische Fehler gefunden werden, bei denen sich der Benutzer nicht an die gedachten Anweisungen hielt. So konnte bei der

Erstellung eines Dokuments ein Fehler ausgelöst werden, der zu einem ungültigen Dokument führt. Der Fehler wurde ausgelöst, falls der Nutzer keine Typ-variable bei einem Number/Textfeld angegeben hatte. Der Fehler wurde in der 4. Iteration behoben.

Neben diesen Tests versuchte die Bezugsperson ihr erstes richtiges Formular zu erstellen und dies ohne Anleitung. Aus diesem ersten Test konnte ich einige Schlussfolgerungen ziehen. Die erste Feststellung war, dass er generell versuchte längere Fragen einzugeben als wir im Moment in der Datenbank erlaubten. Eine Einschränkung die schnell behoben werden konnte. Gleichzeitig fiel bei diesen ersten Tests auf, dass dem Ersteller weitere Möglichkeiten zum Testen gegeben werden mussten als die Vorschaufunktion.

3.5 Vierte Iteration

Für die vierte und letzte Iteration war mein Ziel möglichst viele der aufgetauchten Probleme zu berichtigen, einige der technischen Schulden zu tilgen [9] und als wichtigster Punkt eine Überarbeitung des Prozesses für die Erstellung der Formulare zu implementieren.

3.5.1 Anpassung des Formular Erstellungssystems

Die Benutzertests (siehe Abschnitt 3.4.4) zeigten, dass für neue Formulare eine ausführliche Bewertungsfunktion fehlte. Zu häufig wurden Fehler erst entdeckt, wenn das erste Formular ausgefüllt wurde und nicht bei der Vorschau. Um diesen Prozess zu erweitern, war es meine Idee einen neuen Status für jedes Dokument einzuführen. In diesem Status sollte nur der Ersteller das Formular benutzen können und das wie jedes veröffentlichte. Dies gewährleistet, dass ein Formular ordentlich getestet wird, bevor es für andere Nutzer sichtbar ist. Der Ersteller hat also die zusätzliche Verantwortung ebenfalls das erstellte Formular zu testen.
Schwieriger wäre es die semantischen und syntaktischen Fehler der Fragen zu beheben. Um diese beseitigen zu können kann jeder Administrator auf einer neuen Seite die Frage eines Attributes bearbeiten.

3.5.2 Refactoring

Um ein weiteres Bearbeiten zu erleichtern, mussten einige Funktionen angepasst werden (siehe Abschnitt 3.4.3). Einige der komplexeren Funktionen hatten eine Länge von über 100 Zeilen mit einigen Teilblöcken, die sich leicht auslagern lassen. Ebenfalls wäre es nötig einige der Namen anzupassen. Im Verlauf der Iterationen haben sich einige Namen geändert. So wurde z.B. aus Besuch die Untersuchung. Ich sah ein hohes Gefahrenpotential diese Namen wenige Tage vor der Abgabe zu verändern. Deswegen erschien mir die Umbenennung eher als erste wichtige Aufgabe für die Zeit danach.(vgl. Abschnitt 4)

4 Fazit und Ausblick

Am Ende der Bearbeitungszeit konnte ich eine Seite vorweisen, die die Anforderungen erfüllt. Alle Funktionalitäten waren gegeben.
Die iterative Entwicklung erwies sich als nützlich. Sie verhinderte eine falsche Priorisierung und gleichzeitig, dass Anforderungen komplizierter interpretiert wurden, als sie gefordert waren.
Eine Betrachtung der Frameworks sprachübergreifend wäre eine bessere Wahl gewesen, als diese zuerst durch die Wahl der Sprache zu limitieren. Zu früh sortierte ich so beliebte Frameworks aus. Frameworks wie Django(Python), Ruby on Rails(Ruby) und Catalyst(Perl) hätten es verdient, genauer betrachtet zu werden. Ich bin mir jedoch sicher, dass ich keine andere Wahl, im Rahmen dieser Arbeit, getroffen hätte. Codeigniter ermöglichte eine einfache erste Auseinandersetzung mit einem MVC Framework. Die Aussicht für ein PHP Framework einen Nachfolger zu finden ist ebenso hoch.
Mit dem Fortschritt der Seite sammelten sich einige Ideen bei denen ich mir bewusst war, dass diese innerhalb des Zeitrahmens nicht bearbeitet werden konnten.

Hier seien einige genannt:
1. Es muss einen höheren Nutzerstatus als Administrator geben, dessen zusätzliche Aufgabe es ist, unerwünschten Administratoren ihre Rechte wieder zu nehmen und unbrauchbare, veröffentlichte Formulare zu löschen.
2. Eine Passwort Wiederherstellungsfunktion
3. Die Anwendung müsste ebenfalls gegen Mittelsmannangriffe geschützt werden. Hierfür müsste ein SSL-Zertifikat erstellt werden.
4. Die Fehlerausgabe bearbeiten, so dass sie die Beschreibung ausgibt und nicht den Variablennamen, damit es für den Nutzer eindeutig wird.
5. Ein automatische, wöchentliche Sicherheitskopie der Datenbank
7. Die Implementation einer EAV Lösung für den Abschluss spezieller Formulare.
8. Eine einheitliche Namensgebung der Variablen, Funktionen und Tabellen.
9. Außerdem sollten ausgiebige Softwaretests durchgeführt werden.

Auch komplexere Aufgaben könnten die Anwendung voranbringen. Eine mögliche Fortführung wären Usability-Tests mit mehreren Personen der Zielgruppe als Fokusgruppe und entsprechende Anpassungen anhand der Resultate.
Eine andere Ausbaumöglichkeit bietet das Werkzeug, zum erstellen der Formulare. Dieses wäre mit Komfortfunktionen, wie z.B. Drag & Drop Elementen und einer Echtzeitvorschau angenehmer zu benutzen. Ebenfalls könnten zusätzliche Funktionen hinzugefügt werden. Eine Möglichkeit wären individuelle Validierungsregeln für jedes Attribut, oder die Festlegung von primären Attributen des Krankheitsblocks, die in der Tabellenansicht angezeigt werden.

Literaturverzeichnis

[1] SP Arlt and P Haimerl. Cystic ovaries and ovarian neoplasia in the female dog – a systematic review. *Reproduction in Domestic Animals*, 51:3–11, 2016.

[2] Valentin Dinu and Prakash Nadkarni. Guidelines for the effective use of entity-attribute-value modeling for biomedical databases. *International journal of medical informatics*, 2006.

[3] Lisa Lancor and Samyukta Katha. Analyzing php frameworks for use in a project-based software engineering course. In *Proceeding of the 44th ACM Technical Symposium on Computer Science Education*, SIGCSE '13, pages 519–524, New York, NY, USA, 2013. ACM.

[4] Dr. Matthias Leul. Check point firewall-1 / vpn-1 ng ai check point ngx. http://www.fw-1.de/krypto/angriffe.html Checked 15.07.2017.

[5] University of Liverpool. Data from veterinary practices: Real-time syndromic surveillance. https://www.liverpool.ac.uk/savsnet/about/savsnet-vet/ Checked 15.06.2017.

[6] University of London. About. http://www.rvc.ac.uk/vetcompass/about Checked 01.08.2017.

[7] British Columbia Institute of Technology. Codeigniter 3: Userguide. https://www.codeigniter.com/userguide3/general/welcome.html.

[8] Steve R. Palmer and Mac Felsing. *A Practical Guide to Feature-Driven Development*. Pearson Education, 1st edition, 2001.

[9] Robert L. Nord Philippe Kruchten and Ipek Ozkaya. Technical debt: From metaphor to theory and practice. https://www.computer.org/csdl/mags/so/2012/06/mso2012060018.pdf.

[10] w3techs. Usage statistics and market share of php for websites. https://w3techs.com/technologies/details/pl-php/all/all Checked 01.08.2017.

A Anhang

Draft of a Template for „Reprocases"

Male dogs suffering cryptorchidism

Practrice

Name of the vet: _____ Country: _____

Specialisation: none nationalboard specialization Dipl. ECARI Dipl. ACT

Male Dog

Breed_____ Age ____ Years ____ Months

Housing in house kennel

First visit

Right testicle Location scrotal inguinal abdominal ectopic

Size (measured via ultrasound): ____ diameter ____ length

Notes: _____

Left testicle Location scrotal inguinal abdominal ectopic

Size (measured via ultrasound): ____ diameter ____ length

Notes: _____

Treatment

hCG (500 IU i.m. 4 times in a two week period)

Buserelin (120 ng/kg i.m. 4 times in a two week period)

Abbildung 4: Cryptorchidism

erste Seite Cryptorchidism

no treatment

massageperformed

Second visit

Timeafter lasttreatment: ____ days

Right testicle location scrctal inguiral abdominal ectopic

Size (measured via ultrasound):____ diameter____ length

Notes: _____

Left testicle location scrctal inguiral abdominal ectopic

Size (measured via ultrasound):____ diameter____ length

Notes: _____

Third visit

Timeafter lasttreatment: ____ days

Right testicle location scrctal inguiral abdominal ectopic

Size (measured via ultrasound):____ diameter____ length

Notes: _____

Left testicle location scrctal inguiral abdominal ectopic

Size (measured via ultrasound):____ diameter____ length

Notes:_____

Abbildung 5: Cryptorchidism2
zweite Seite Cryptorchidism

Draft of a Template for „Reprocases"

Induction of ovulation in female bitches

Practice

Name of the vet: _____ Country: _____

Specialisation: none nationalboard specialization Dipl. ECARI Dipl. ACT

Female dog

Breed_____ Age ____ Years ____ Months

Housing in house kennel

Female reproductive history

Interval of heats: _____ months

Litters: _____

Female medical history

Diseases: diabetes incontinence pyometra

Treatments for contraception: progesteronederivates aglepristone PGF

Visit (at time or close to ovulation induction)

Day in heat (day 1 = first day with observation of sanguineous discharge observed by the owner): _____

Vagina: Discharge yes no

Quality of discharge clear semisanguineous sanguineous pus

Vaginal folds (vaginal inspection via speculum) ____ (-/+/++/+++)

Vaginal smear: basalcells____%

Parabasalcells___%

Abbildung 6: Ovulationp1
erste Seite induction of ovulation

Intermediate cells ____%

Superficial cells ____%

Potatoe chips ____%

WBC none single many

Bloodtest results: Oestradiol _____ pmol/L

Progesterone _____ nmol/L

LH _____ nmol/L

Treatment

Day in heat (day 1 = first day with observation of sanguineous discharge observed by the owner): _____

hCG (500 IU i.m.)

Buserelin (120 ng/kg i.m.)

other treatment protocol: _____

Outcome

Pregnancy

yes

No

Number of puppies identified via ultrasound: _____

Number of puppies born alive: _____

Observed side effects that are likely due to ovulation induction:

_____

_____

_____

Abbildung 7: ovulationp2
zweite Seite induction of ovulation

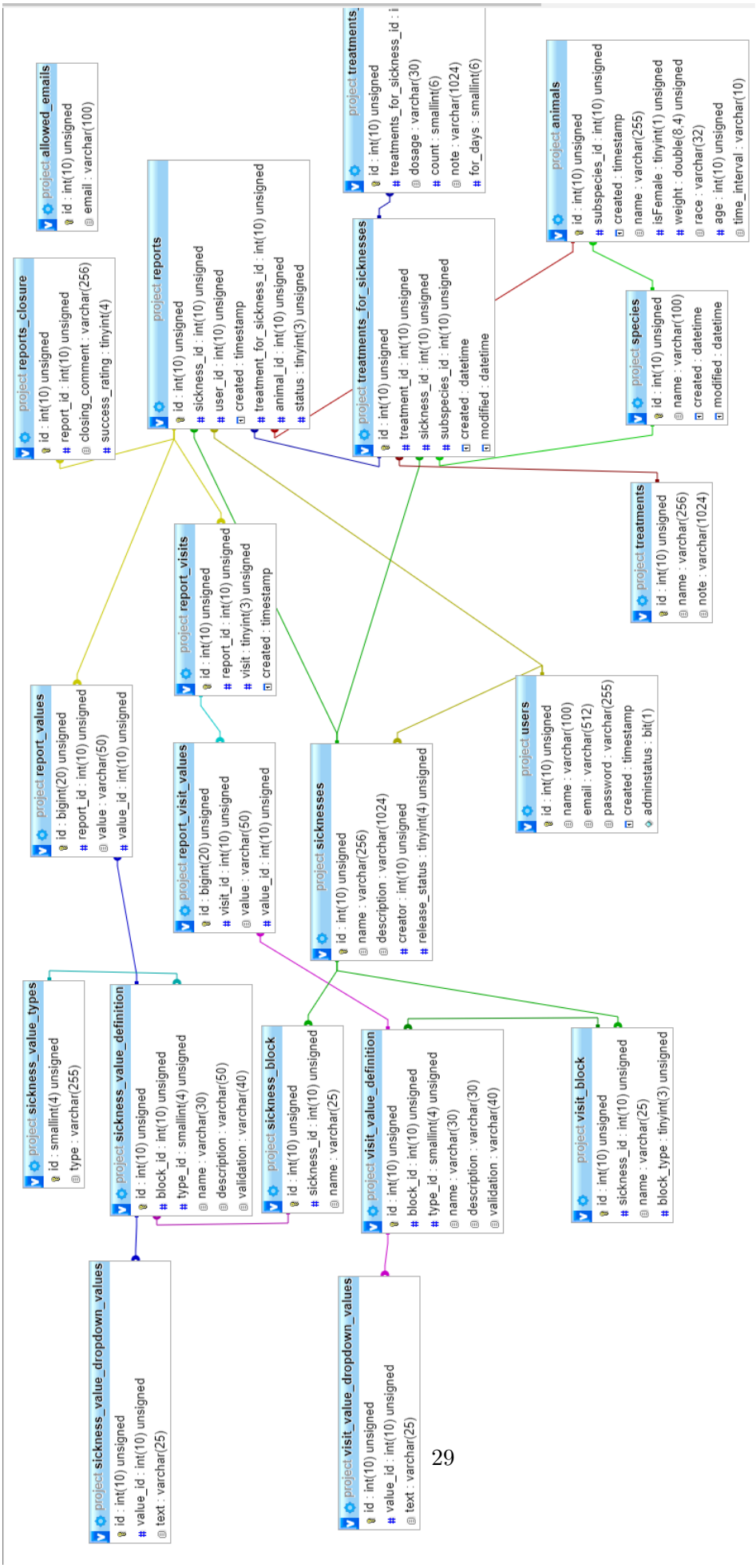

Abbildung 8: Database