



Diplomarbeit

# Communicational

**Evaluation eines Ticketsystems für den Landesschulrat Tirols**

Peter Pollheimer      Jakob Tomasi      Elias Gabl

Imst, 27. Juni 2017

Betreut durch:

Stefan Stolz

Alexander Scharmer

# Eidesstattliche Erklärung

Ich erkläre an Eides statt, dass ich die vorliegende Diplomarbeit selbst verfasst und keine anderen als die angeführten Behelfe verwendet habe. Alle Stellen, die wörtlich oder inhaltlich den angegebenen Quellen entnommen wurden, sind als solche kenntlich gemacht. Ich bin damit einverstanden, dass meine Arbeit öffentlich zugänglich gemacht wird.

---

Ort, Datum

---

Jakob Tomasi

---

Peter Pollheimer

---

Elias Gabl

# Abnahmeerklärung

Hiermit bestätigt der Auftraggeber, dass das übergebene Produkt dieser Diplomarbeit den dokumentierten Vorgaben entspricht. Des Weiteren verzichtet der Auftraggeber auf unentgeltliche Wartung und Weiterentwicklung des Produktes durch die Projektmitglieder bzw. die Schule.

---

Ort, Datum

---

Auftraggeber

# Vorwort

Beauftragt wurde das Projektteam von Helmut Hammerl im Namen des Landeshochschulrates Tirol. Der Kontakt mit Helmut Hammerl wurde von Alexander Scharmer hergestellt. In einer ersten Besprechung erläuterte Helmut Hammerl die Problemstellung, ein Ticketsystem verwenden zu müssen, welches auf Mobilgeräten kaum effektiv eingesetzt werden kann. Das Projektteam und Helmut Hammerl einigten sich darauf, im Rahmen des Diplom- und Abschlussprojektes des Teams dessen Dienste in Anspruch zu nehmen und im Gegenzug die Projektbetreuung zu übernehmen.

# Kurzfassung

Ein bereits bestehendes Ticketsystem des Landesschulrats Tirol soll durch Änderungen funktional vereinfacht und optisch aufbereitet werden. Mit diesem System können IT-ManagerInnen Probleme melden, die die IT Infrastruktur einer Schule betreffen. Nach den ersten abgeschlossenen Phasen des Projektes, wurde jedoch vom Projektteam festgestellt, dass der Umfang des vorgegebenen Ticketsystems zu umfangreich ist.

Um die Projektziele zu erreichen und Änderungen zu bewerkstelligen, liegt der Fokus auf einer vom Projektteam verfassten Dokumentation um mit dem vorgegebenen Quellcode bestmöglichst zu arbeiten.

# Abstract

As of today, the education authority of Tyrol uses the open source ticketing system OSTicket. It enables IT-Managers to report problems with school's IT infrastructure. The scope of this project is to simplify the usage of OSTicket by creating a new web-based interface and removing unused functions. However, while getting used to the internal works of OSTicket, the team realized it was way too crufty.

In order to meet the project's requirements, it became necessary to find an alternative system, which enables the team to deliver a ticketing system capable of adaption in an efficient manner.

# Zusammenfassung

Communicational erweitert ein bestehendes System des Landesschulrats Tirol, basierend auf OSTicket. OSTicket ist ein Webbasiertes Open-Source Ticketsystem. Es wird verwendet, um IT-ManagerInnen an Tirols Schulen Probleme ihrer IT-Infrastruktur an die Zuständigen SystemadministratorInnen bekannt zu geben. Dies erfolgt unter der Domain itsys-tirol.at. Dieses Portal unterstützt nur Geräte mit großen Bildschirmen (ab 1024 Pixel Bildschirmbreite). Dies erschwert die Verwendung mit mobilen Geräten erheblich. Durch die im Rahmen dieses Projektes vorgenommenen Anpassungen an der Benutzeroberfläche wird die Benutzung mit verschiedensten Endgeräten wie z.B. Smartphones und Tablets ermöglicht.

Des Weiteren befindet der Projektpartner - OStR. Prof. Mag. Hammerl Helmut - die Grundkonfiguration von OSTicket für zu voluminös. Deshalb sollen möglichst viele nicht verwendete Funktionen im angepassten User-Interface weggelassen werden, um die Bedienbarkeit und Userfreundlichkeit zu erhöhen.

Nach vielen Stunden der Einarbeitung in OSTicket wurde dem Projektteam klar, dass dieses durch seine allgemeine Beschaffenheit wie die fehlende Dokumentation des Codes und die Vermischung von Programmlogik und HTML-Elementen schlecht für die vom Projektteam geplante Anpassungen geeignet ist. Aufgrund dessen wurde eine Alternative gesucht, mit der die im Rahmen des Projektes durchzuführenden Änderungen effizienter zu bewerkstelligen sind. Gefunden wurde ein Fork von OSTicket mit dem Namen Katak (eine Abspaltung, die von einem anderen Entwicklerteam durchgeführt und gewartet wird), welcher für die Projektbedürfnisse eine hervorragende Alternative darstellt.

# Inhaltsverzeichnis

<b>1. Projektmanagement</b>	<b>11</b>
1.0.1. Team . . . . .	11
1.0.2. Betreuer . . . . .	11
1.0.3. Partner . . . . .	11
1.0.4. Ansprechpartner . . . . .	11
1.0.5. Risikoanalyse . . . . .	12
1.1. Pflichtenheft . . . . .	14
1.2. IST Zustand . . . . .	14
1.3. SOLL Zustand . . . . .	15
1.3.1. SMART . . . . .	15
1.3.2. Produkteinsatz und Umgebung . . . . .	16
1.3.3. Projektumfeldanalyse . . . . .	16
1.3.4. Stakeholder . . . . .	17
1.3.5. Funktionalitäten . . . . .	17
1.3.6. Muss Anforderungen . . . . .	17
1.3.7. Soll Anforderungen . . . . .	18
1.3.8. Testszenarien und Testfälle . . . . .	18
1.4. Testfälle . . . . .	18
1.4.1. Testfall A . . . . .	18
1.4.2. Testfall B . . . . .	19
1.4.3. Testfall C . . . . .	19
1.5. Planung . . . . .	20
1.5.1. Meilensteine . . . . .	20
1.5.2. Gant-Chart . . . . .	20



<b>2. Produktvorstellung</b>	<b>21</b>
2.1. Realisierbarkeit mit OSTicket . . . . .	21
2.1.1. Codeausschnitte osTicket . . . . .	23
2.1.2. Evaluation Katak . . . . .	26
2.1.3. Evaluation OSTicky . . . . .	27
2.2. Systemdokumentation . . . . .	28
2.2.1. OSTicket . . . . .	28
2.2.2. Bootstrap . . . . .	30
2.2.3. PHP . . . . .	31
2.2.4. MySQL . . . . .	32
2.3. Werkzeuge . . . . .	32
2.3.1. NetBeans . . . . .	34
2.3.2. XAMPP Apache . . . . .	35
2.3.3. MySQL Workbench . . . . .	36
2.4. Modelle . . . . .	38
2.4.1. Klassendiagramm . . . . .	38
2.5. Implementierung . . . . .	39
2.6. Datenbankentwurf . . . . .	41
2.6.1. Tabellenbeschreibung der Datenbank von OSTicket . . . . .	41
2.6.2. Beschreibung der Datenbankanbindung von OSTicket . . . . .	58
2.6.3. Tabellenbeschreibung der Datenbank von unserem Java EE Prototyp . . . . .	66
2.6.4. Beschreibung der Datenbankanbindung von unserem Java EE Prototyp . . . . .	72
<b>3. Problemanalyse</b>	<b>77</b>
3.1. USE-Case-Analyse . . . . .	77
3.1.1. Ablaufbeschreibung . . . . .	82
3.2. User-Interface-Design . . . . .	85
3.3. Prototyp . . . . .	86
<b>4. Systementwurf</b>	<b>88</b>
4.1. Technologie . . . . .	88
4.2. Architektur . . . . .	89
4.3. Benutzerschnittstellen . . . . .	90
4.4. Klassenentwurf . . . . .	93

4.5. Akzeptanztests . . . . .	94
<b>5. Projektevaluation</b>	<b>95</b>
5.1. Einführung . . . . .	95
5.2. Planungsabweichungen . . . . .	95
5.2.1. Projektpartner & -betreuer . . . . .	96
5.3. Zusammenarbeit . . . . .	96
5.3.1. Arbeitsaufteilung Projektteam . . . . .	96
<b>6. Zusammenfassung</b>	<b>97</b>
<b>Abbildungsverzeichnis</b>	<b>98</b>
<b>Tabellenverzeichnis</b>	<b>99</b>
<b>Quelltexte</b>	<b>101</b>
<b>Literaturverzeichnis</b>	<b>102</b>
<b>A. Anhang-Kapitel</b>	<b>103</b>
A.1. Anhang-Section . . . . .	103

# **1. Projektmanagement**

**1.0.1. Team**

**1.0.2. Betreuer**

**1.0.3. Partner**

**1.0.4. Ansprechpartner**

## 1.0.5. Risikoanalyse

### Risikoidentifikation

Folgende Risiken können während der Projektdurchführung erwartet werden:

- R1** Projektmitglied steigt aus dem Projekt aus.
- R2** Projektpartner stellt seine Kooperation ein.
- R3** Projektpartner ändert seine Anforderungen.
- R4** Termine können nicht eingehalten werden.
- R5** Anforderungen werden nicht erreicht.

### Bewertung und Behandlung

Die aufgelisteten Risiken werden nach Auswirkung und Eintrittswahrscheinlichkeit bewertet. Je höher die Zahl in der Tabelle, desto höher ist die Auswirkung bzw. Einwirkung.

Risiko	Wahrscheinlichkeit	Auswirkung
<b>R1</b> Ausstieg Projektmitglied	3	10
<b>R2</b> Projektpartner stellt Kooperation ein	2	10
<b>R3</b> Projektpartner ändert seine Anforderungen	4	7
<b>R4</b> Termine können nicht eingehalten werden	3	6
<b>R5</b> Anforderungen werden nicht erreicht	2	8

Tabelle 1.1.: Analyse Einwirkung & Auswirkung

## Risikomatrix

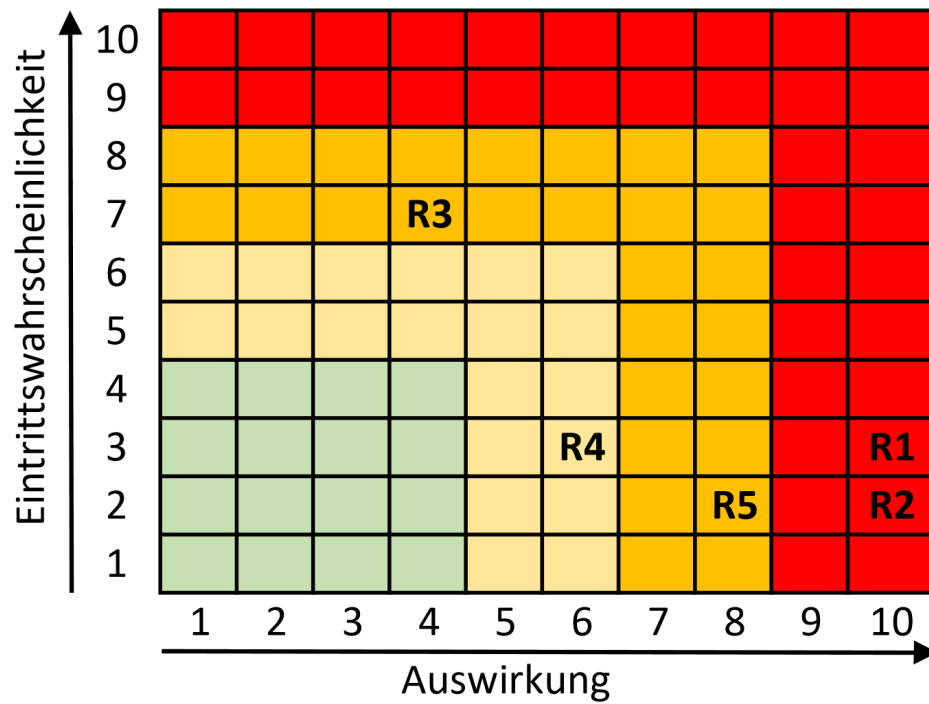


Abbildung 1.1.: Risikomatrix

## 1.1. Pflichtenheft

## 1.2. IST Zustand

IT-ManagerInnen an Tirols Schulen können Probleme mit der Infrastruktur melden und Anfragen zur Beschaffung von Ressourcen/Komponenten stellen.

Sie können den Bearbeitungsverlauf ihrer Tickets beobachten. SystembetreuerInnen empfangen die Tickets der IT-ManagerInnen, welche sich in ihrem Cluster befinden. SystembetreuerInnen bearbeiten die Tickets und antworten auf die Anfragen.

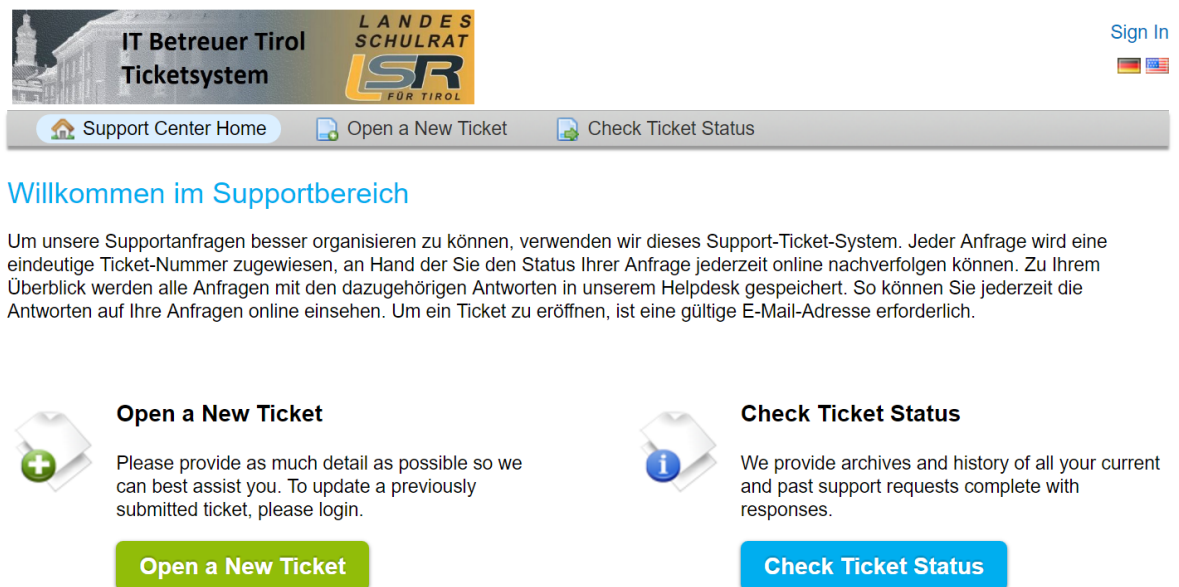


Abbildung 1.2.: IST-Zustand OS-Ticket

Die Benutzerschnittstelle ist derzeit nur auf die Benutzung mit großen Bildschirmen (ab 1024 Pixel Bildschirmbreite) ausgelegt. Sie kann sich nicht an kleinere Formate (Smartphone, etc.) anpassen.

## **1.3. SOLL Zustand**

Bessere Usability soll mit Hilfe von Mobile-First Orientierung auf Basis von Bootstrap erreicht und wenn möglich die Ticketerstellung vereinfacht werden.

Das Backend soll die Aufteilung in mehrere hierarchische Organisationseinheiten ermöglichen und eine Erweiterung von Landesebene auf Bundesebene zulassen. Des Weiteren gilt es, den AnwenderInnen den Ticketingprozess intuitiver zu gestalten.

### **1.3.1. SMART**

#### **S** Spezifisch

Systembetreuer und IT-Manager können Support- und Beschaffungsanfragen mit Hilfe des Ticketsystems abwickeln.

#### **M** Messbar

Systembetreuer empfangen die Tickets und kümmern sich um die Probleme. Die Schulen werden in Cluster eingeteilt und von Systembetreuern verwaltet.

#### **A** Attraktiv

Die Plattform muss auf jedem Endgerät verfügbar sein (Responsive Design). Das Absetzen und Ansehen von Tickets soll vereinfacht werden, die Plattform bietet einige Funktionen die für das System relevant sind.

#### **R** Realisierbar

Zum Realisieren wird eine Testumgebung von Seiten des Betreuers zur Verfügung gestellt. Das Responsive Design wird mithilfe eines Framework (Bootstrap) realisiert.

#### **T** Terminisierbar

Im Juni 2017 wird das Projekt abgeschlossen und eine technische Dokumentation des Projekts liegt vor.

### **1.3.2. Produkteinsatz und Umgebung**

### **1.3.3. Projektumfeldanalyse**

#### **Einflussfaktoren**

Das Projekt wurde durch den Landesschulrat Tirol in Auftrag gegeben. Die Ansprechperson, Herr Helmut Hammerl, informiert uns über den IST und SOLL-Zustand der Plattform und unterstützt das Projekt mit Ideen und Hilfestellungen bei Problemstellungen.

Des Weiteren beeinflussen die Anwender (IT-Manager) und die Systembetreuer der Plattform das Projektergebnis. Da auf die Anwenderfreundlichkeit viel Wert gelegt wird, spielen diese Faktoren eine wirkliche Rolle.

Die Projektbetreuer Stefan Stolz und Alexander Scharmer sind Ansprechpartner für Fragen, die bezüglich Problemstellungen während des Projekts auftreten können, und damit enorm einflussreich.



### 1.3.4. Stakeholder

#### Stakeholder Identifizieren

Stakeholder	Einfluss	Konfliktpotential
Helmut Hammerl	3	0
Dr. Stefan Walch	2	0
Stefan Stolz	1	+
Alexander Scharmer	1	+
Michael Gamper	0	0
LSI DI Anton Lendl	3	0
Team Mitglieder	2	0

Tabelle 1.2.: Stakeholder Identifikation

#### Stakeholder Klassifizieren

Stakeholder	Risiken durch Stakeholder	Strategien
Mag. Helmut Hammerl	Ändern der Ansprüche	Unterschriebenes Pflichtenheft
Dr. Stefan Walch	Nicht bestätigen des Projektantrages	Durchdachter Projektantrag
Stefan Stolz	falsche Informationen, kein Interesse	regelmäßiges Treffen
Alexander Scharmer	falsche Informationen, kein Interesse	regelmäßiges Treffen
Michael Gamper	Nicht bestätigen des Projektantrages	Durchdachter Projektantrag
LSI DI Anton Lendl	Nicht bestätigen des Projektantrages	Durchdachter Projektantrag
Team Mitglieder	Mangelnde Motivation	Faire Arbeitsverteilung

Tabelle 1.3.: Stakeholder Klassifikation

### 1.3.5. Funktionalitäten

### 1.3.6. Muss Anforderungen

IT-ManagerInnen und Systembetreuer müssen sich unter [itsys-tirol.at](http://itsys-tirol.at), einem Portal des Landesschulrates, basierend auf OSTicket anmelden können. Des Weiteren sollen die Schulen selbst bestimmen, wer einen Zugang zum Portal erhalten soll, um Tickets erstellen zu können.

Die angemeldeten IT-ManagerInnen müssen Probleme mit der Infrastruktur melden können und Anfragen zur Beschaffung von Ressourcen bzw. Komponenten einreichen können.

SystembetreuerInnen müssen die Tickets der IT-ManagerInnen empfangen, welche sich in ihrem Cluster befinden. Die eingereichten Tickets sollen von den Systembetreuern bearbeitet werden können.

### **1.3.7. Soll Anforderungen**

Es soll eine neue Weboberfläche entwickelt werden, die auf dem HTML & CSS Framework Bootstrap basiert. Dieses soll sich auf Einfachheit in der Anwendung und Benutzerfreundlichkeit fokussieren. Die Latenzzeit sollte so niedrig wie möglich gehalten werden um ein Reibungsloses Arbeiten zu ermöglichen.

### **1.3.8. Testszenarien und Testfälle**

Die Testfälle in unserem Projekt beziehen sich auf das Ticketingsystem OSTicket.

## **1.4. Testfälle**

### **1.4.1. Testfall A**

- **Beschreibung:** Ein IT-Manager möchte ein Ticket erstellen.
- **Vorbedingung:** Der User benötigt ein internetfähiges Gerät und muss im Portal eingeloggt sein.
- **Aktion:** Der Benutzer wählt den Tab "Neues Ticket" und füllt die notwendigen Felder aus.

- **Soll-Reaktion:** Das System setzt das Ticket für den zuständigen Systembetreuer sichtbar.

Tester:

M Datum:

### **1.4.2. Testfall B**

- **Beschreibung:** Ein Systembetreuer möchte ein Ticket bearbeiten.
- **Vorbedingung:** Der Systembetreuer benötigt ein internetfähiges Gerät und muss im Portal eingeloggt sein.
- **Aktion:** Der Systembetreuer wählt den Tab "Meine Tickets" und wählt ein Ticket aus das bearbeitet werden muss. Durch die Beschreibung des Tickets, weiß der Systembetreuer über die Problemstellung Bescheid und kann dementsprechend handeln.
- **Soll-Reaktion:** Der Systembetreuer kann sich um die Problemstellung kümmern und das Ticket nach erfolgreicher Bearbeitung wieder schließen.

Tester:

Datum:

### **1.4.3. Testfall C**

- **Beschreibung:** Ein Anwender möchte ein bestimmtes Ticket suchen und dieses begutachten.
- **Vorbedingung:** Der Anwender benötigt ein internetfähiges Gerät und muss im Portal eingeloggt sein.
- **Aktion:** Der Anwender gibt im Suchfeld ein Stichwort ein nach dem er suchen möchte.
- **Soll-Reaktion:** Das gesuchte Ticket soll angezeigt werden.

Tester:

Datum:

## **1.5. Planung**

### **1.5.1. Meilensteine**

- 15.10.2016 Testumgebung wurde unseren Bedürfnissen angepasst, Planungsphase abgeschlossen.
- 15.02.2017 Die Grundfunktionen können dem Auftraggeber präsentiert werden.
- 18.05.2017 Die einzelnen Alternativen wurden evaluiert und bewertet.
- 25.06.2017 Dokumentation fertiggestellt

### **1.5.2. Gant-Chart**

## 2. Produktvorstellung

In diesem Kapitel wird das Softwaresystem OSTicket dokumentiert bzw. vorgestellt sowie dessen Eignung für die Anforderungen des Projektauftraggebers bearbeitet. Die im Zuge des Projektes geplante Modifikation von OSTicket wird behandelt und es wird erläutert, warum diese nicht in einem angemessenen Rahmen durchführbar ist. Des Weiteren werden die Ansätze und unternommenen Schritte um das ursprüngliche Projektziel zu erreichen, dargestellt.

### 2.1. Realisierbarkeit mit OSTicket

OSTicket hat sich als umfangreicher herausgestellt wie zu Beginn des Projekts angenommen wurde. Nach Ausarbeitung des konzeptuellen Zieles des Projektes erfolgte eine lange Phase der Evaluation, Einarbeitung und Dokumentation von OSTicket. Nach ca. 30 Stunden dieser (viel zu langen) Phase, in der die Komplexität und Schwerfälligkeit des Systems OSTicket sich nicht bezwingen lassen wollte. Die Anzahl der Dateien setzt sich folgendermaßen zusammen:

Dateityp	Dateianzahl
.php	414
.css	15
.less	9
.sql	61
.html	1
Gesamt	500

Tabelle 2.1.: Verwendete Dateitypen in OSTicket

Ein weiteres Hindernis ergibt sich durch den Mangel an einer auch nur annähernd aktuellen Dokumentation des laufend erweiterten und angepassten OSTicket.

Damit ist OSTicket nicht nur wegen der fehlenden Dokumentation kaum zu bändigen, sondern aufgrund seines ständigen Wachstums zu „crufty“, um mit sinnvollem Aufwand angepasst zu werden:

In the jargon of hackers<sup>1</sup>, it is called „cruft.“ An operating system<sup>2</sup> that has many, many layers of it is described as „crufty.“

Hackers hate to do things twice, but when they see something crufty, their first impulse is to rip it out, throw it away, and start anew. [...] Like an upgrade to an old building, cruft always seems like a good idea when the first layers of it go on – just routine maintenance, sound prudent management. This is especially true if (as it were) you never look into the cellar, or behind the drywall. But if you are a hacker who spends all his time looking at it from that point of view, cruft is fundamentally disgusting, and you can’t avoid wanting to go after it with a crowbar. Or, better yet, simply walk out of the building – let the Leaning Tower of Pisa fall over – and go make a new one THAT DOESN’T LEAN. Stephenson (1999)

Um die Realisierbarkeit zu veranschaulichen nun einige Codeauszüge aus OSTicket.

---

<sup>1</sup>Es darf angenommen werden, dass Neal Stephenson in seinem Essay mit Hacker vor allem Softwareentwickler aus der Open-Source Welt meint.

<sup>2</sup>Neal Stephenson’s Essay bezieht sich vor allem auf Betriebssysteme, der Jargonausdruck findet in der Softwareentwicklung jedoch generelle Anwendung.

### 2.1.1. Codeausschnitte osTicket

Quelltext 2.1: main.inc.php

```
18 #Disable direct access.
19 if(isset($_SERVER['SCRIPT_NAME']))
20 && !strcasecmp(basename($_SERVER['SCRIPT_NAME']),
21 ,basename(__FILE__)))
22 die('kwaheri rafiki!');
23
24 require('bootstrap.php');
25 Bootstrap::loadConfig();
26 Bootstrap::defineTables(TABLE_PREFIX);
27 Bootstrap::i18n_prep();
28 Bootstrap::loadCode();
29 Bootstrap::connect();
30
31 #Global override
32 $_SERVER['REMOTE_ADDR'] = osTicket::get_client_ip();
```

In diesem Codeauszug wird die Lesbarkeit des Codes sehr gut veranschaulicht, der Aufruf von elf statischen Funktionen ohne Dokumentation erschwert die Erweiterbarkeit um ein vielfaches. Der einzige Kommentar des Programmierers *Don't Monkey around with it* ist weder aussagekräftig noch hilfreich.

Quelltext 2.2: pwreset.php

```
51 $inc = 'register.confirmed.inc.php';
52 $acct->confirm();
53 // FIXME:
    The account has to be uncached in order for the lookup
54 // in the ::processSignOn to detect the confirmation
55 ModelInstanceManager::uncache($acct);
56 // Log the user in
57 if ($client =
    UserAuthenticationBackend::processSignOn($errors))
58 {
59     if ($acct->hasPassword() && !$acct->get('backend'))
60     {
61         $acct->cancelResetTokens();
62     }
63     // No password setup yet -- force one to be created
64     else
65     {
66         $_SESSION['_client']['reset-token'] =
        $_GET['token'];
67         $acct->forcePasswdReset();
68     }
```

Auch in der Datei *pwreset.php* ist nur eine sehr dürftige Dokumentation vorzufinden. Es sind lediglich vier Inhaltslose Kommentare wie *makes includes happy* und ein *FIXME* vorzufinden. Diese geben keinerlei Aufschluss über die Funktionalität und Zuständigkeit des Codes.



## Quelltext 2.3: offline.php

```

16 require_once('client.inc.php');
17 if(is_object($ost) && $ost->isSystemOnline())
18 {
19     @header('Location: index.php');
20     //Redirect if the system is online.
21     include('index.php');
22     exit;
23 }
24 $nav=null;
25 require(CLIENTINC_DIR.'header.inc.php');

```

Der einzige Kommentar des Programmierers *modify to fit your needs* erschwert die Lesbarkeit dieser Datei enorm.

## Quelltext 2.4: client.inc.php

```

38 /* include what is needed on client stuff */
39 require_once(INCLUDE_DIR.'class.client.php');
40 require_once(INCLUDE_DIR.'class.ticket.php');
41 require_once(INCLUDE_DIR.'class.dept.php');
42
43 //clear some vars
44 $errors=array();
45 $msg='';
46 $nav=null;
47 //Make sure the user is valid..before doing anything else.
48 $thisclient = UserAuthenticationBackend::getUser();

```

In der Datei client.inc.php wird lediglich dokumentiert, dass diese Datei in jeder client page inkludiert wird. Der Code in dieser Datei besteht hauptsächlich aus statischen Funktionsaufrufen.

### **2.1.2. Evaluation Katak**

Katak, ein Fork<sup>3</sup> von OSTicket, lässt sich laut Angaben der Entwickler leichter installieren, bedienen und anpassen. Das liegt, nach Informationen der Webseite von Katak<sup>4</sup> hauptsächlich an einem vereinfachten Datenbankentwurf und der Absenz einiger weniger PHP-Klassen. Die Möglichkeit einer Anpassung von OSTicket wurde dahingehend ausgeforscht, eine Installation auf einem Testsystem vorzunehmen und gewisse Anpassungen durchzuführen. Die Ergebnisse des Problemlösungsversuch in diese Richtung sind im Abschnitt 2.1 zu finden.

Katak bietet laut Entwickler folgende Vorteile:

- Internationalisierung durch Gettext, Administratoren bestimmen die Sprache des Systems.
- Das Hinzufügen von mehreren Attachments für Tickets.
- Der Überblick über Tickets und dessen Status wurde erweitert. Mitarbeiter können nur Tickets bearbeiten, denen sie auch zugeteilt wurden.
- Passwortsicherheit wurde durch die Verwendung von SHA512<sup>5</sup> verbessert.

Vor allem die Vereinfachung des Codes und die Trennung zwischen der grafischen Oberfläche und der Logik sollen die Arbeit mit diesem Open Source System um ein Vielfaches erleichtern.

---

<sup>3</sup>unabhängige Weiterentwicklung

<sup>4</sup><http://www.katak-support.com/en>

<sup>5</sup>Hashfunktion

### **2.1.3. Evaluation OSTicky**

OSTicky präsentierte sich zu Beginn als vielversprechender Ansatz, den Vorstellungen von Helmut Hammerl und dem Landesschulrat am besten und effizientesten zu entsprechen.

Diese Lösung unterscheidet sich insofern von einer OSTicket Installation und einer Katak Installation, dass es sich bei OSTicky nicht um ein eigenständiges Ticketsystem handelt. Es muss viel mehr wie eine Brücke zwischen einer OSTicket Vanilla Instanz und einem Joomla Interface verstanden werden. Mit OSTicky kann man also ein komplett eigenständiges, unabhängiges Webinterface implementieren, das die Funktionalitäten und Datenbestände von OSTicket utiliziert und somit als Interface für das Ticketsystem fungiert.

Ist bereits ein Joomla System in Verwendung, so würde sich OSTicky als geeignete Lösung herausstellen, da dies jedoch nicht der Fall ist, würde sich der Arbeitsaufwand um ein vielfaches erhöhen.

## 2.2. Systemdokumentation

### 2.2.1. OSTicket

OSTicket ist ein weit verbreitetes, quelloffenes Ticketingsystem. Es verfügt über eine zentrale MySQL-Datenbank und ein Multi-User Web Interface. Dieses Webinterface gilt es im Zuge dieses Projektes so aufzubereiten, dass es neben den üblichen Auflösungen von Desktop- und Laptopbildschirmen auch Mobile Displays wie die von Smartphones oder Tablets unterstützt.

Es wird eine Minimum-Impact Strategie verfolgt, das heißt dass im Zuge der Projektdurchführung die Änderungen auf das Vanilla<sup>6</sup>-System so gering als möglich ausfallen. Damit wird einerseits das Ziel verfolgt, eine gewisse Übersichtlichkeit zu bewahren, was in einer kompakten Dokumentation resultieren soll.

Des weiteren sollen nicht benötigte Funktionalitäten bewahrt bleiben, um systeminterne Reibungen zu vermeiden und Funktionen im Nachhinein mit geringem Aufwand verwenden zu können.

Abbildung 2.1.: OSTicket Logo



#### Möglichkeiten mit OSTicket

OSTicket ist sehr anpassbar. Ticketformulare und Formularfelder lassen sich mit geringem Aufwand anpassen, je nach den spezifischen Anforderungen des Helpdesks. Sollte einmal ein wahrer Sturm an Tickets eingehen, ist auch das kein Problem. Tickets lassen sich simpel priorisieren und filtern. Noch mehr Übersicht bringt die

---

<sup>6</sup>Von „Vanilla“ spricht man, wenn ein System unangegriffen und sich in dessen Auslieferungszustand befindet.

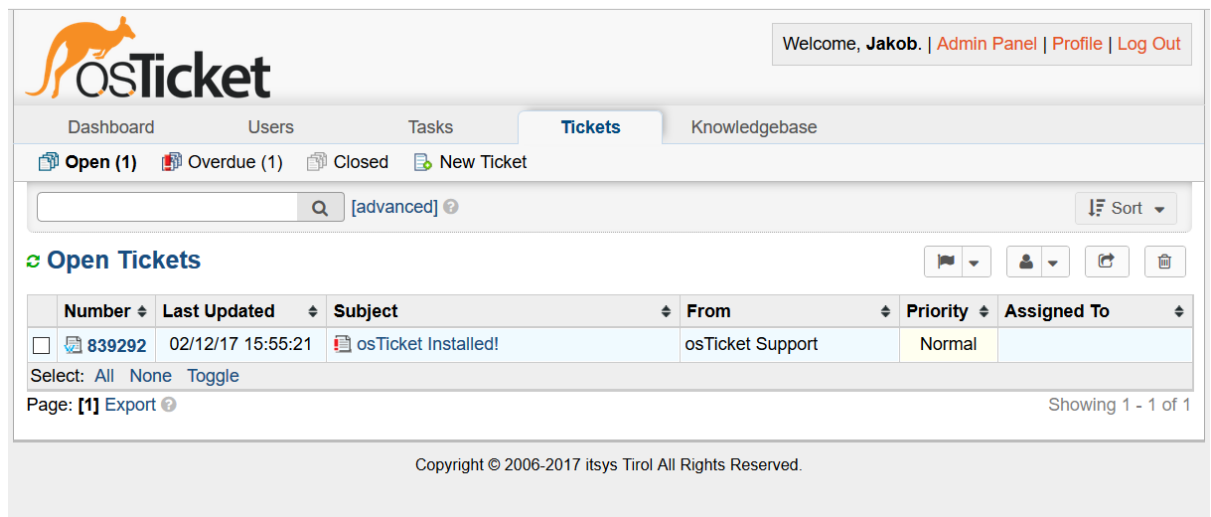


Abbildung 2.2.: Die standard-Weboberfläche für Administratoren

Möglichkeit, Tickets in Hilfsthematiken einzustufen. Sollten mehrere Supportmitarbeiter gleichzeitig Arbeiten, geschieht dies ohne Problem: Ticketkollisionen werden automatisch mit Sperrvariablen vermieden.

### Warum OSTicket?

OSTicket findet bereits an einigen Schulen in Tirol Anwendung. Dies soll, laut initialem Wunsch des Projektauftraggebers, im Allgemeinen dem Landesschulrat für Tirol und im Speziellen Projektbetreuer Helmut Hammerl, beibehalten und auf weitere Schulen ausgeweitet werden.

Die Vorteile dieser Verbreitung und der damit einher gehenden Standardisierung kann für die Schulen große Vorteile mit sich bringen. Die bis jetzt verwendeten Systeme sind sehr unterschiedlich und in keiner Weise miteinander kompatibel. OSTicket soll als anwenderfreundliches System diese Schwierigkeiten vermeiden und beseitigen. Dadurch soll eine bessere Kommunikation zwischen Systembetreuern und IT Managern an Tirols Schulen ermöglicht werden.

### 2.2.2. Bootstrap

Das Framework Bootstrap wird verwendet, um eine Anwender freundliche Mobile-first Oberfläche zu erstellen. Bootstrap bietet umfangreiche Gestaltungsvorlagen wie Formulare, Buttons, Tabellen und weitere nützliche Oberflächengestaltungssysteme.



Abbildung 2.3.: Twitter Bootstrap Logo

Gründe die für die Verwendung von Bootstrap sprechen:

- Responsive-Design
- Hohe Kompatibilität bezüglich Browser
- Übersichtliche Dokumentation
- Eine Vielzahl von Templates ist verfügbar

#### Bootstrap

Compiled and minified CSS, JavaScript, and fonts. No docs or original source files are included.

[Download Bootstrap](#)

#### Source code

Source Less, JavaScript, and font files, along with our docs. **Requires a Less compiler and some setup.**

[Download source](#)

#### Sass

[Bootstrap ported from Less to Sass](#) for easy inclusion in Rails, Compass, or Sass-only projects.

[Download Sass](#)

Abbildung 2.4.: Bootstrap ist ein vielseitiges und anpassbares Framework

### 2.2.3. PHP

OSTicket basiert auf der Skriptsprache PHP. PHP ist ein rekursives Akronym und bedeutet "Hypertext Preprocessor". PHP kann direkt in HTML<sup>7</sup> eingebettet werden. Die Skriptsprache bietet eine Vielzahl an Funktionen Out Of The Box, wie zum Beispiel das Senden und Empfangen von Emails, Manipulation einer Datenbank oder das Befüllen einer Webseite mit Inhalten.



Abbildung 2.5.: PHP Ver. 7 Logo

PHP ist der etablierte Standard, wenn es um serverseitige Webprogrammierung geht. Serverseitige Webprogrammierung findet statt, wenn ein Benutzer mit seinem Browser<sup>8</sup> eine Webseite ansurft. Es wird eine sogenannte HTTP-Anfrage an den Webserver (welcher unter der Webseitenadresse erreichbar ist) gesendet. Dieser analysiert die Anfrage (welche Seite der Benutzer sehen will) und befüllt das HTML-Grundgerüst mit dynamischen Inhalten. Dann wird die Seite über das Internet zurück zum Browser des Benutzers übertragen und dargestellt.

Ein großer Vorteil von PHP ist, dass es für Einsteiger in die Programmierung gut zu erlernen ist, dennoch aber äußerst flexibel, um auch umfangreiche Anwendungen (Wie OSTicket) umsetzen zu können.

Die Flexibilität von PHP zeigt sich auch in der Freiheit, es auf vielen unterschiedlichen Betriebssystem wie Microsoft Windows, Apple OSX und einer Vielzahl an Linuxdistributionen verwenden zu können. Auch allerhand Webserver verstehen PHP wie ihre Muttersprache: Apache, IIS und nginx, um nur einige zu nennen.

---

<sup>7</sup>Hypertext Markup Language, definiert das Grundgerüst einer Webseite

<sup>8</sup>Software zum Benutzen des World Wide Web; Bspw. Microsoft Internet Explorer, Mozilla Firefox

#### **2.2.4. MySQL**

MySQL ist ein Datenbanksystem das weltweit verbreitet ist. Es gibt eine Open-Source-Software aber auch eine kommerzielle Enterpriseversion.



Abbildung 2.6.: MySQL Logo

MySQL wird für die Datenspeicherung von Webservices verwendet. Es wird meistens zusammen mit dem Apache Webserver und der Skriptsprache PHP verwendet. Dies war auch der Grund weshalb wir MySQL verwenden da OSTicket in der Skriptsprache PHP geschrieben ist. MySQL und die Bibliotheken sind hauptsächlich in C und C++ geschrieben. Dies ist zurückzuführen auf die Erzielung einer guten Performance. MySQL sieht grundsätzlich eine MySQL-Server vor auf dem mehrere Datenbanken erstellt werden können. In einer Datenbank können auch mehrere Tabellen erstellt werden.

### **2.3. Werkzeuge**

Die im Zuge der Projektdurchführung verwendeten Werkzeuge eignen sich für das Projekt aus verschiedenen Gründen.

Ein wichtiger Punkt war die Vertrautheit mit den Werkzeugen: Ein Großteil dieser war uns bereits bekannt beziehungsweise vertraut. Des Weiteren ist die große Community im Internet sehr hilfreich bei auftretenden Problemen und Fragestellungen; meist ist die Lösung innerhalb weniger Minuten gefunden. Da es sich um weit ver-



breitete Entwicklungsumgebungen handelt konnte auch in der Zukunft (während der Zeit der Projektdurchführung) mit der Weiterentwicklung und Verbesserung dieser Tools gerechnet werden.

### 2.3.1. NetBeans

Die integrierte Entwicklungsumgebung NetBeans ist in Java geschrieben und bietet vielerlei Funktionen wie einen umfangreichen Editor, eine passable Codevervollständigung, Code Analysetools, Debugger und kann mit hilfreichen Plug-Ins wie beispielsweise EasyUML, einem Programm zum Erstellen von UML-Klassendiagrammen aus einem vorhandenen Projekt, erweitert werden. Wie in der Abbildung 2.7 zu sehen ist, kann das Farbthema der Entwicklungsumgebung angepasst werden. Das schont die Augen und erhöht den Workflow.

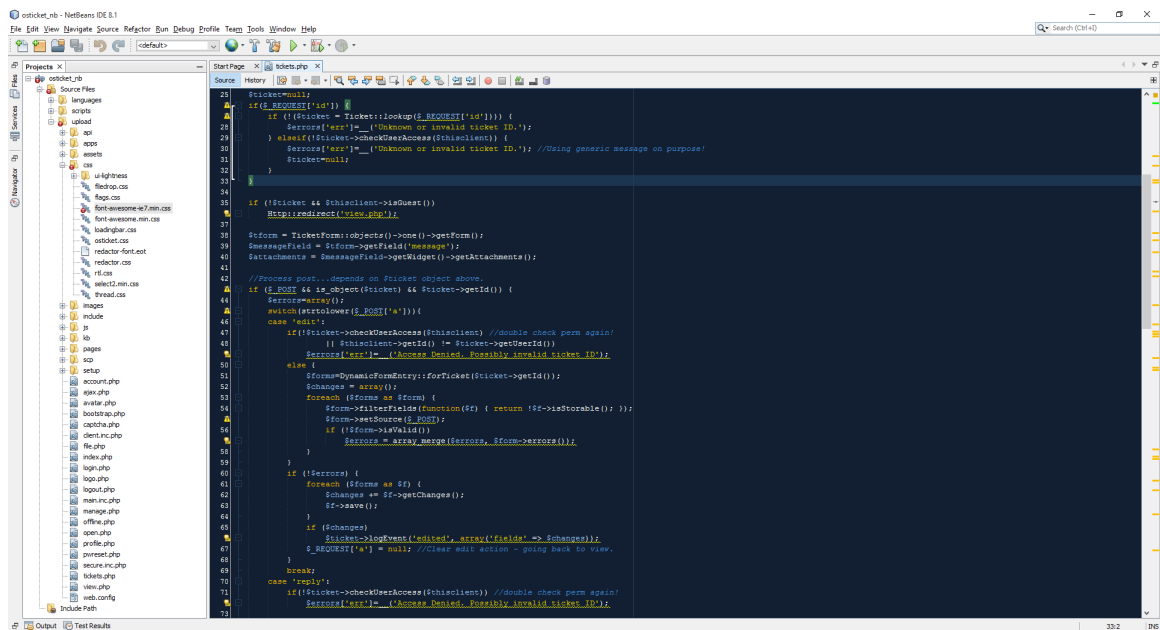


Abbildung 2.7.: Netbeans User Interface

NetBeans unterstützt Programmiersprachen wie Java, PHP, C/C++ und andere. Des Weiteren ist NetBeans plattformunabhängig - wir als Projektteam haben somit die freie Wahl unter welchem Betriebssystem gearbeitet werden soll - die Kompatibilität unserer Ergebnisse ist jederzeit gegeben.

Jedes neue Release wird auch von der Community getestet und evaluiert. Somit ist auch eine Hilfestellung bei Problemen gegeben.

Abbildung 2.8.: Netbeans IDE Logo



### 2.3.2. XAMPP Apache

XAMPP ist eine Zusammenführung von freier Software. Es ermöglicht einfache Installation und Konfiguration des Webserver Apache und der Datenbank MariaDB. Weitere Anwendungen geliefert unter XAMPP sind die FTP-Server ProFTPD oder FileZilla Server, der Mailserver Mercury, der webbasierte Datenbankclient php-MyAdmin und andere offene Software.

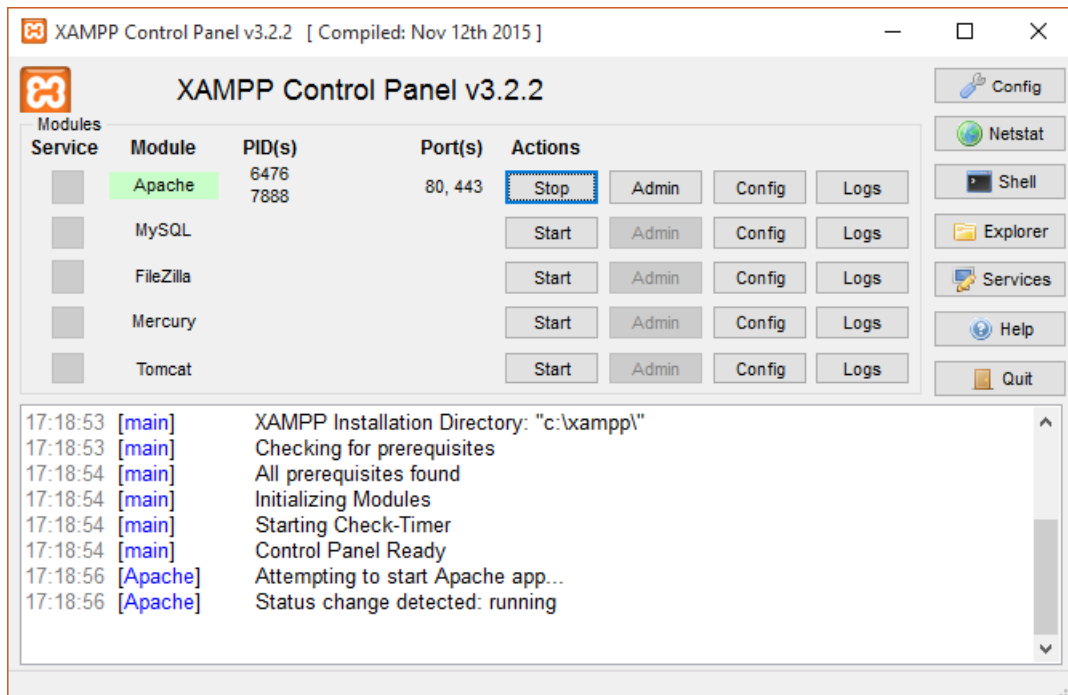


Abbildung 2.9.: XAMPP Control Panel

XAMPP ermöglicht eine einfache und schnelle Installation einer Testumgebung. Da XAMPP ausschließlich für Testumgebungen gedacht ist, sind standardmäßige Sicherheitseinstellungen nicht ausreichend und ist somit nicht für Produktivumgebungen geeignet.

Abbildung 2.10.: XAMPP Logo



### 2.3.3. MySQL Workbench

Die Verwaltung der verwendeten MySQL Datenbank erfolgt mit dem Tool MySQL Workbench. Die vielen Überwachungs- und Konfigurationsmöglichkeiten, die die GUI<sup>9</sup> bietet, macht es möglich, auf aufwändige SQL-Skripte zu verzichten. Damit ist eine einfache Verwaltung und ein erleichterter Umgang mit dieser umfangreichen Datenbank gegeben.

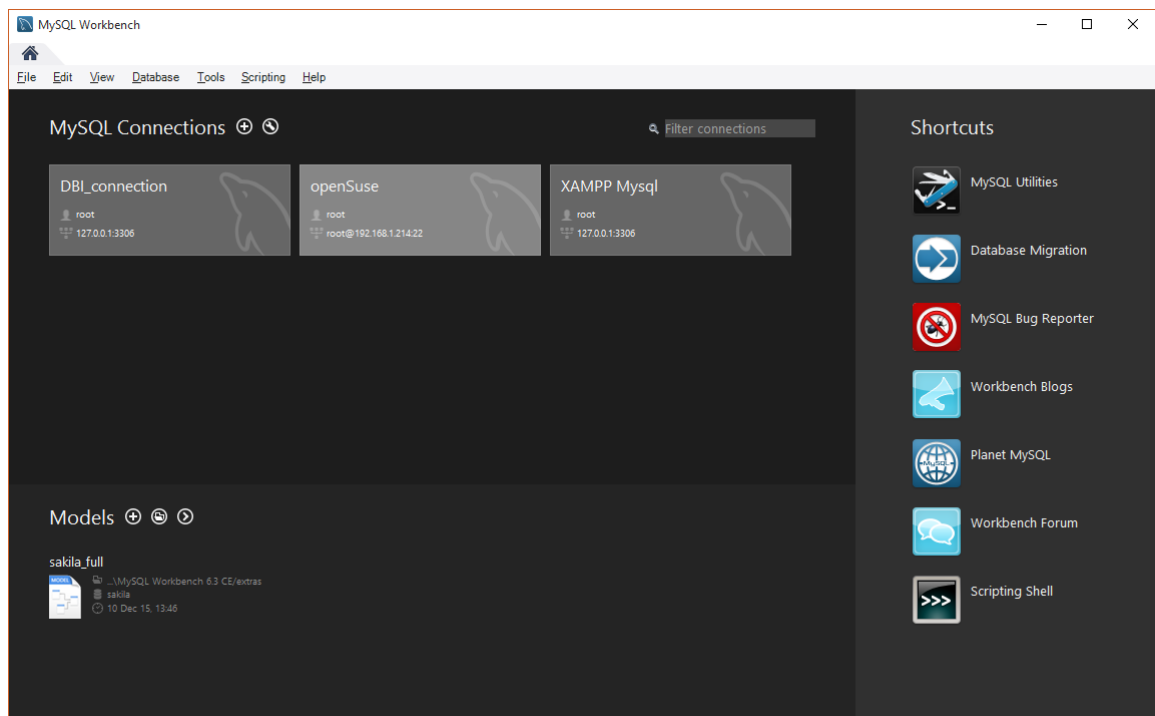


Abbildung 2.11.: MySQL Workbench Start Screen

Auch die Unterstützung von Projektbetreuern und weiteren Lehrpersonen kann in

---

<sup>9</sup>Graphical User Interface

Betracht gezogen werden, da die MySQL Workbench im Unterricht oft Verwendung findet.

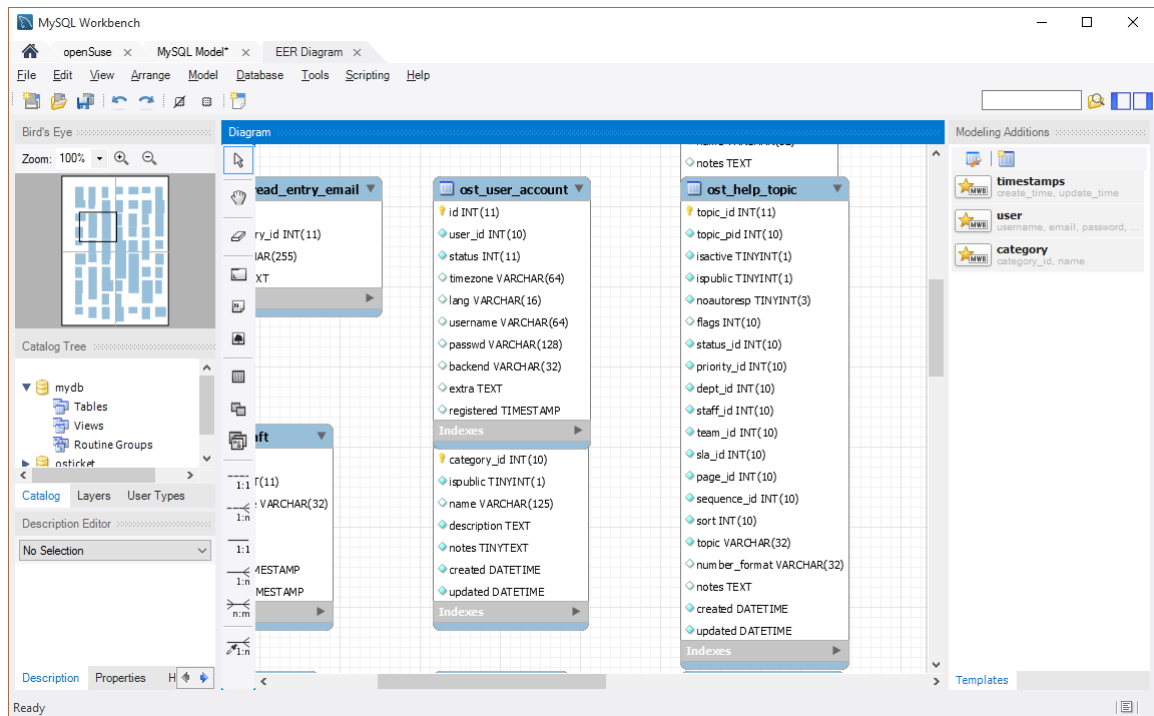


Abbildung 2.12.: MySQL Workbench - Datenbankdiagramm (Ausschnitt)

MySQL Workbench kann neben der Ausführung von Konfigurations- und Datenmanipulationsbefehlen auch Modellierung: Mit dem Integrierten UML Editor kann eine Datenbank (und die zugehörigen SQL Befehle) generiert werden. Auch die entgegengesetzte Richtung ist möglich: Mit wenigen Klicks kann ein UML Diagramm aus einer bestehenden Datenbank erstellt werden. Abbildung 2.12 zeigt ein solches Klassendiagramm im Detail.

## 2.4. Modelle

### 2.4.1. Klassendiagramm

Abbildung 2.13 soll die Klassen welche unter dem Ticketsystem Verwendung finden umreiend beschreiben. Das Diagramm dokumentiert nicht den kompletten systematischen Aufbau des Ticketsystems in seinem "Vanilla-Zustand"<sup>10</sup>, sondern ist eine Zusammenfassung der wichtigsten Klassen, die Anwendung finden und soll lediglich veranschaulichen, welche die fundamentalsten Klassen sind und in welchem Zusammenhang diese stehen.

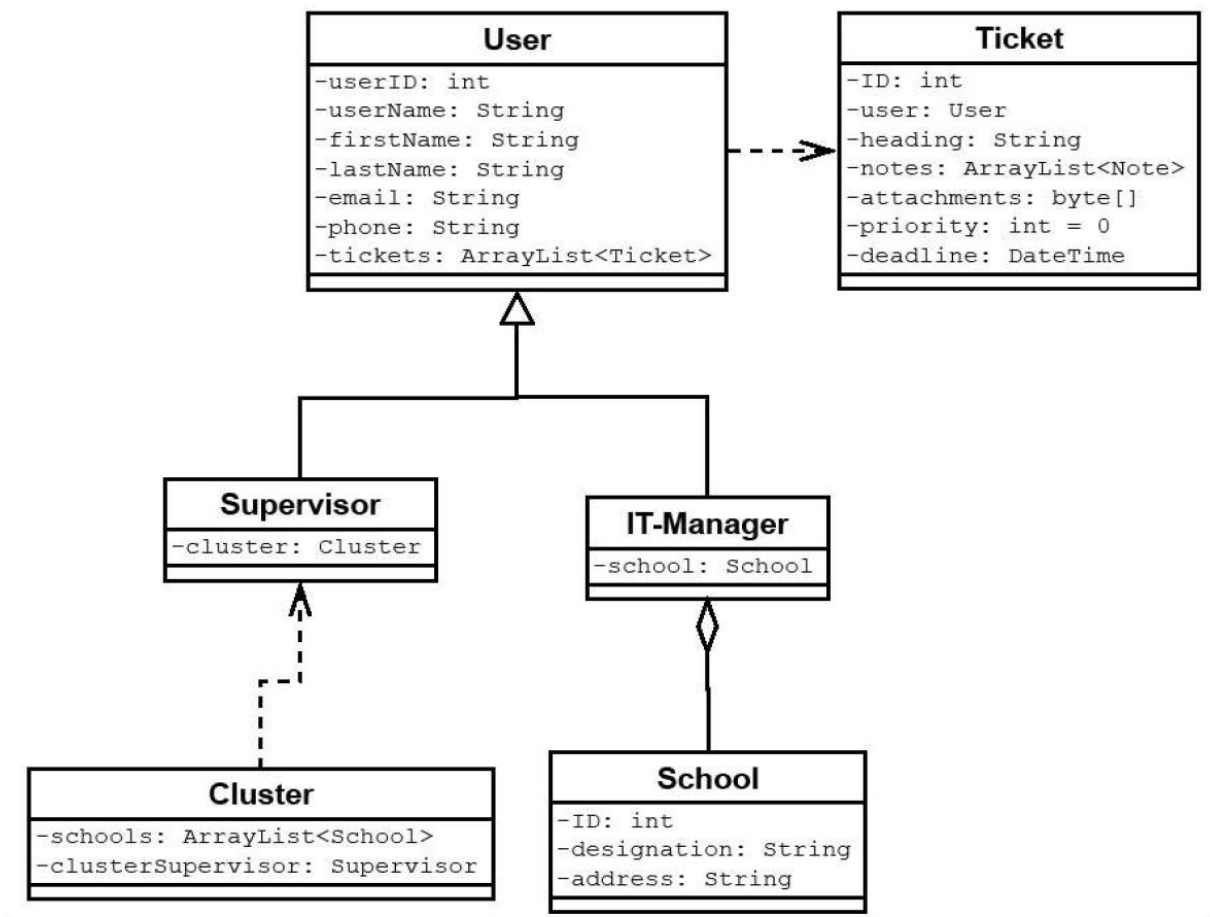


Abbildung 2.13.: Komprimiertes Klassendiagramm

<sup>10</sup>"Vanilla"beschreibt ein System oder Produkt, welches in keinsten Weise verndert (customized) worden ist.

## 2.5. Implementierung

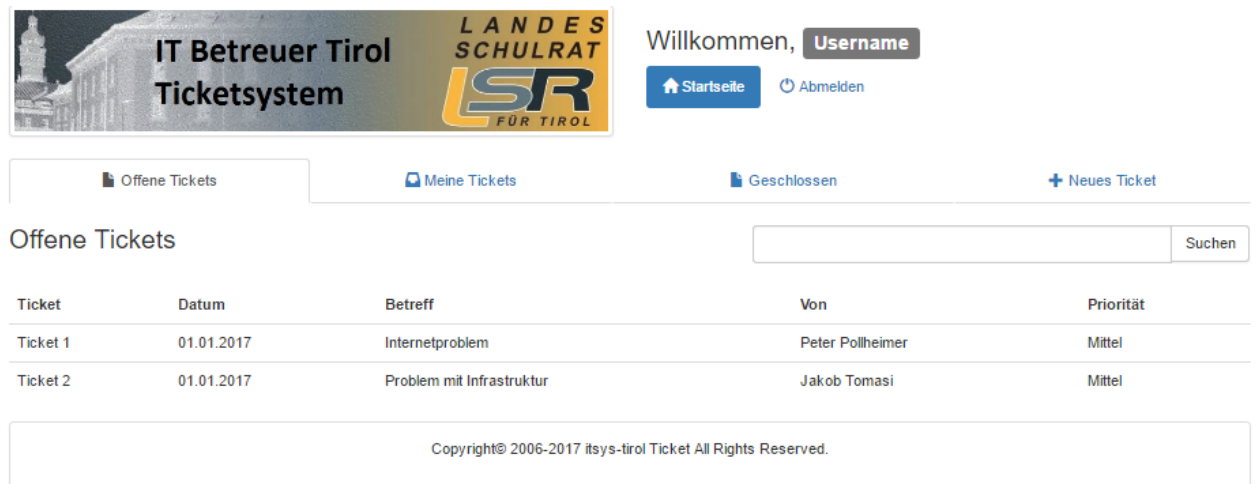
Es wurden zwei GUI Prototypen mit Bootstrap<sup>11</sup> erstellt. Die grafische Oberfläche wurde stark vereinfacht und mit Helmut Hammerl besprochen.

The screenshot displays the 'IT Betreuer Tirol Ticketsystem' web interface. At the top, there is a header banner with the system name and the 'LANDES SCHULRAT LSR FÜR TIROL' logo. To the right of the banner, a user is logged in, indicated by 'Willkommen, Username' and buttons for 'Startseite' and 'Abmelden'. Below the header, a navigation bar contains links for 'Offene Tickets', 'Meine Tickets', 'Geschlossen', and a button for '+ Neues Ticket'. The main form area is titled 'Hilfethema:' and features a dropdown menu currently set to 'Allgemeine Anfrage'. Below this is a 'Betreff:' label followed by a text input field. The 'Details der Anfrage:' section contains a larger text area for the ticket description. Underneath, the 'Priorität:' is set with radio buttons for 'Niedrig', 'Mittel', and 'Hoch'. At the bottom of the form are two buttons: 'Absenden' and 'Abbrechen'. A footer bar at the very bottom contains the copyright notice: 'Copyright© 2006-2017 itsys-tirol Ticket All Rights Reserved.'

Abbildung 2.14.: Ticket erstellen

---

<sup>11</sup>Ein Framework, Kapitel 2.2.2



IT Betreuer Tirol Ticketsystem

LANDES SCHULRAT LSR FÜR TIROL

Willkommen, **Username**

[Startseite](#) [Abmelden](#)

[Offene Tickets](#) [Meine Tickets](#) [Geschlossen](#) [+ Neues Ticket](#)

Offene Tickets

Ticket	Datum	Betreff	Von	Priorität
Ticket 1	01.01.2017	Internetproblem	Peter Pollheimer	Mittel
Ticket 2	01.01.2017	Problem mit Infrastruktur	Jakob Tomasi	Mittel

Suchen

Copyright© 2006-2017 itsys-tirol Ticket All Rights Reserved.

Abbildung 2.15.: Ticketverwaltung

Die Verwaltung der Tickets sollte möglichst einfach wie in der Abbildung 2.15 abgewickelt werden. Offene, geschlossene und Tickets die der eingeloggte User erstellt hat, sollen ausgewählt werden können.



## **2.6. Datenbankentwurf**

### **2.6.1. Tabellenbeschreibung der Datenbank von OSTicket**

In dieser Dokumentation finden Sie eine grobe Beschreibung der Datenbank des Systems OS Ticket. OS Ticket ist ein Ticketsingsystem, das für einfache Support Anwendung entwickelt worden ist. Die Datenbank besteht aus 59 Tabellen, von diesen sind manche nicht mehr aktuell bzw. werden nicht mehr gebraucht, sind aber dennoch vorhanden. Daher werden in dieser Dokumentation nur die wichtigsten Tabellen der Datenbank beschrieben.

In gewissen Spalten ist es leider nicht möglich den Inhalt anzugeben, da es keine gute Dokumentation der Datenbank gibt und man den Inhalt der Spalten aus dem Kontext erschließen muss.

An dieser Stelle sollte ein Datenmodell stehen, aber da jenes von OSTicket recht groß und komplex ist, hat es hier nicht Platz. Sie finden es auf dem mitgelieferten Datenträger.(Der Dateiname ist: OSTicketDB.mwb).

## ost\_config

In dieser Tabelle werden wichtige Informationen über das System OSTicket gespeichert. Die Werte werden mit einem Schlüssel in der Datenbank abgelegt (Key und Value). Diese Tabelle hat keine Referenz zu anderen Tabellen.

Column Name:	Datatype:	Content:
id	INT(11)	Eindeutg ID der Information. Dieser Wert ist rein technisch und hat neben der Eindeutigkeit keine weitere Aussagekraft.
namespace	VARCHAR(64)	
key	VARCHAR(64)	Key der Information um die Suche zu erleichtern
value	TEXT	Informations Inhalt
updated	DATETIME	Erstelldatum der Information

Tabelle 2.2.: tab:ost-config

**ost\_attachment**

Sämtliche Informationen über den Anhang eines Tickets finden sich in dieser Tabelle. Beinhaltet sind der Datentyp, der Name, und die id des Anhanges.

Column Name:	Datatype:	Content:
id	INT(10)	Eindeutige ID des Anhanges. Dieser Wert ist rein technisch und hat neben der Eindeutigkeit keine weitere
object_id	int(11)	Referenz auf ein Object das dem Anhang übergeben wird
type	CHAR(1)	Datentyp des Anhanges
file_id	int(11)	Referenz auf das File das sich im Anhang befindet
name	VARCHAR(255)	Name des Anhanges
inline	TINYINT(1)	Beschreibt den Zitierstil des Anhangs
lang	VARCHAR(16)	Die Sprache in der, der Anhang verfasst wurde

Tabelle 2.3.: tab:ost-attachment

## ost\_canned\_response

In dieser Tabelle werden vorgefertigte Antworten für Tickets abgelegt. Über die Antwort wird der Titel, die Sprache, der Inhalt, das Erstelldatum und das Änderungsdatum gespeichert.

Column Name:	Datatype:	Content:
canned_id	INT(10)	Eindeutige ID der Nachricht. Dieser Wert ist rein technisch und hat neben der Eindeutigkeit keine weitere Aussagekraft.
dept_id	int(10)	Referenz auf die Abteilungen die diese Nachricht verwenden können
isenabled	TINYINT(1)	Gibt an ob die Nachricht freigegeben ist oder nicht.
titel	VARCHAR(255)	Titel der Nachricht
response	TEXT	Text den die Nachricht beinhaltet
lang	VARCHAR(16)	Sprache der Nachricht
notes	TEXT	Anmerkung zur Nachricht
created	DATETIME	Erstelldatum der Nachricht
updated	DATETIME	Änderungsdatum der Nachricht

Tabelle 2.4.: tab:ost-canned-response

**ost\_content**

In dieser Tabelle werden Inhalte der Seiten aufgenommen. Es wird angegeben, ob der Inhalt aktiv ist oder nicht. Jeder Inhalt umfasst auch einen Titel und einen Body. Zusätzlich werden noch der Typ und das Erstell- /Änderungsdatum angegeben.

Column Name:	Datatype:	Content:
id	INT(10)	Eindeutige ID des Contents. Dieser Wert ist rein technisch und hat neben der Eindeutigkeit keine weitere Aussagekraft.
isactive	TINYINT(1)	Gib an ob der Inhalt aktiv ist oder nicht
type	VARCHAR(32)	Gib den Typ vom Inhalt an
name	VARCHAR(255)	Name vom Inhalt
body	TEXT	Der Bodyinhalt des Seiten Content
notes	TEXT	Anmerkungen zum Content
created	DATETIME	Erstelldatum des Content
updated	DATETIME	Änderungsdatum des Content

Tabelle 2.5.: tab:ost-content

## ost\_department

Diese Tabelle speichert Informationen über eine Abteilung. Jede Abteilung hat einen Namen und eine Signatur. Ein Feld, das angibt, ob die Abteilung öffentlich ist oder nicht. Zudem wird festgehalten, zu welcher Gruppe die Abteilung gehört und wann diese erstellt worden ist.

Column Name:	Datatype:	Content:
id	INT(10)	Eindeutige ID der Abteilung. Dieser Wert ist rein technisch und hat neben der Eindeutigkeit keine weitere Aussagekraft.
pid	INT	Referenz zu der Tabelle ost_plugin
tpl_id	INT(10)	Referenz zum verwendeten Template für die Abteilung
sla_id	INT(10)	Referenz zur verwendeten Sla Vorlage
email_id	INT(10)	Referenz zu der Verwendeten E-Mail der Abteilung
autores_email_id	INT(10)	
manager_id	INT(10)	Referenz zum User der zum Manager der Abteilung ernannt wurde
flags	INT(10)	
name	VARCHAR(128)	Name der Abteilung
signature	TEXT	Signatur der Abteilung
ispublic	TINYINT(1)	Gib an ob die Abteilung öffentlich sichtbar ist
group_membership	TINYINT(1)	Gib an zu welcher Gruppe die Abteilung gehört
ticket_auto_response	TINYINT(1)	Das vordefinierte Standartticket der Abteilung

Tabelle 2.6.: tab:ost-department

message_auto_response	TINYINT(1)	Die vordefinierte Standardantwort der Abteilung auf Tickets
path	VARCHAR	Pfad der Abteilung
updated	INT(10)	Änderungsdatum der Abteilung
created	INT(10)	Erstelldatum der Abteilung

Tabelle 2.7.: tab:ost-department2

## ost\_faq

In dieser Tabelle werden die am häufigsten gestellten Fragen und die jeweiligen Antworten dazu gespeichert. Des Weiteren werden noch Schlüsselwörter und Anmerkungen zur Frage erfasst.

Column Name:	Datatype:	Content:
faq_id	INT(10)	Eindeutige ID der Frage. Dieser Wert ist rein technisch und hat neben der Eindeutigkeit keine weitere Aussagekraft.
category_id	INT(10)	Referenz zu der Kategorie der die Frage zugeordnet ist
ispublished	TINYINT(1)	Gib an ob die Frage öffentlich abrufbar ist oder nicht
question	VARCHAR(255)	Die Frage
answer	TEXT	Die Antwort zur Frage
keywords	TINYTEXT	Die Schlüsselwörter der Frage
notes	TEXT	Anmerkung zur Frage
created	DATETIME	Erstelldatum der Frage
updated	DATETIME	Änderungsdatum der Frage

Tabelle 2.8.: tab:ost-faq

## ost\_staff

Diese Tabelle enthält sämtliche Informationen über die Mitarbeiter\_innen einer Firma, welche das OSTicket- System verwenden. Es werden unter anderem der Vorname, der Nachname, der Username, das Passwort, die Email-Adresse, die Telefonnummer, die Sprache, ob der/die Mitarbeiter\_in aktiv ist und ob er/sie ein Administrator ist, gespeichert. Die Tabelle enthält auch Informationen über die Abteilung und die Rolle des/der Mitarbeiter\_in.

Column Name:	Datatype:	Content:
staff_id	INT(11)	Eindeutige ID des Mitgliedes. Dieser Wert ist rein technisch und hat neben der Eindeutigkeit keine weitere Aussagekraft.
dept_id	INT(10)	Referenz zur Abteilung des Mitgliedes
role_id	INT(10)	Referenz zur Rolle des Mitgliedes
username	VARCHAR(32)	Der Username des Mitgliedes
firstname	VARCHAR(32)	Der Vorname des Mitgliedes
lastname	VARCHAR(32)	Der Nachname des Mitgliedes
passwd	VARCHAR(128)	Das Passwort des Mitgliedes in Hash Form
backend	VARCHAR(32)	
email	VARCHAR(128)	Die Email-Adresse des Mitgliedes
phone	VARCHAR(24)	Die Telefonnummer des Mitgliedes
phone_ext	VARCHAR(6)	Referenz zur Email an die, die Info zum Ticketeingang geschickt wird
mobile	VARCHAR(24)	Die Handynummer des Mitgliedes
signature	TEXT	Die Signatur des Mitgliedes

Tabelle 2.9.: tab:ost-staff



lang	VARCHAR(16)	Die Sprache des Mitgliedes
timezone	VARCHAR(64)	Die Zeitzone in der sich das Mitglied befindet
locale	VARCHAR(16)	Wo sich das Mitglied genau befindet(Ort, Stadt, Land)
notes	TEXT	Anmerkungen zum Mitglied
isactive	TINYINT(1)	Gib an ob das Mitglied aktiv ist
isadmin	TINYINT(1)	Gib an ob das Mitglied ein Administrator ist
isvisible	TINYINT(1)	Gib an ob das Mitglied für andere sichtbar ist
onvacation	TINYINT(1)	Gib an ob der Mitarbeiter im Urlaub ist
assigned_only	TINYINT(1)	
show_assigned_tickets	TINYINT(1)	Hat das Mitglied zugeteilte Tickets
changed_passwd	TINYINT(1)	Gib an ob das Mitglied das Passwort schon mal geändert hat
max_page_size	INT(11)	Gibt die maximale Anzahl an Tickets an die auf der Übersichtsseite des Mitgliedes angezeigt werde
auto_refresh_rate	INT(10)	Gib die Taktrate an wie oft die Übersichtsseite des Mitgliedes automatisch aktualisiert werden soll

Tabelle 2.10.: tab:ost-staff2

default_signature_type	ENUM('none', 'mine', 'dept')	Gib die default Signatur bei der Beantwortung eines Tickets an
default_paper_size	ENUM('Letter', 'Legal', 'Ledger', 'A4', 'A3')	Gib das default Format der Antwort auf ein Ticket an
extra	TEXT	Hier können zusätzliche Informationen über das Mitglied gespeichert werden
permissions	TEXT	Die Zugriffsrechte des Mitgliedes
created	DATETIME	Erstelldatum des Mitgliedes
lastlogin	DATETIME	Datum an dem sich das Mitglied zuletzt angemeldet hat
passwdreset	DATETIME	Datum vom letzten rücksetzen des Passwortes
updated	DATETIME	Änderungsdatum des Mitgliedes

Tabelle 2.11.: tab:ost-staff3

## ost\_ticket

Diese Tabelle enthält sämtliche Informationen über ein Ticket. Dazu gehören die id eines Users, welcher das Ticket abgesetzt hat, die Ticket- Erkennungsnummer, die Email-Adresse des Absenders, das Thema des Tickets und wer es bearbeiten muss. Außerdem gibt es ein Feld, das speichert, ob auf das Ticket schon geantwortet wurde, wann es erstellt und gegebenenfalls verändert wurde und ob es schon geschlossen wurde.

Column Name:	Datatype:	Content:
ticket_id	INT(11)	Eindeutige ID des Tickets. Dieser Wert ist rein technisch und hat neben der Eindeutigkeit keine weitere Aussagekraft.
number	VARCHAR(20)	Ticket Erkennungsnummer
user_id	INT(11)	Referenz zum User der das Ticket abgesetzt hat.
user_email_id	INT(11)	Referenz zur Email-Adresse des Users der das Ticket abgesetzt hat
status_id	INT(10)	Referenz zum Status des Tickets
dept_id	INT(10)	Referenz zur Abteilung der das Ticket zugewiesen wurde
sla_id	INT(10)	Referenz zum Sla des Tickets
topic_id	INT(10)	Referenz zum Thema dem das Ticket zugeordnet worden ist
staff_id	INT(10)	Referenz zur Tabelle ost_staff
team_id	INT(10)	Referenz zum Team dem das Ticket zugewiesen worden ist
email_id	INT(10)	Referenz zur Email an die, die Info zum Ticketeingang geschickt wird
lock_id	INT(10)	Referenz zur Tabelle ost_lock
flags	INT(10)	

Tabelle 2.12.: tab:ost-ticket

ip_address	VARCHAR(64)	Die IP-Adresse von der das Ticket abgesetzt worden ist
source	ENUM(Web, Email, Phone, API, Other)	
source_extra	VARCHAR(40)	
isoverdue	TINYINT(1)	Gib an ob der Bearbeiter überfällig mit der Bearbeitung ist
isanswered	TINYINT(1)	Gib an ob dem Absender des Tickets schon geantwortet wurde
duedate	DATETIME	Gib an bis wann das Ticket bearbeitet sein sollte
est_duedate	DATETIME	Bis wann es geplant ist das Ticket bearbeitet zu haben
reopened	DATETIME	Gib an wann ein geschlossenes Ticket zuletzt geöffnet worden ist
closed	DATETIME	Speichert das Datum an dem das Ticket geschlossen worden ist
lastupdate	DATETIME	Gib das letzte Änderungsdatum an
created	DATETIME	Erstelldatum des Tickets
updated	DATETIME	Änderungsdatum des Tickets

Tabelle 2.13.: tab:ost-ticket2

**ost\_user**

Hier werden jene User gespeichert, die nicht zur den Mitarbeiter\_innen gehören. Diese User können im OS-Ticketsystem lediglich Tickets abschicken. Es werden der Name, der Status und wann das Ticket erstellt worden ist, gespeichert. Des Weiteren kann der User auch einer Organisation zugeordnet werden.

<b>Column Name:</b>	<b>Datatype:</b>	<b>Content:</b>
id	INT(10)	Eindeutige ID des Users. Dieser Wert ist rein technisch und hat neben der Eindeutigkeit keine weitere Aussagekraft.
org_id	INT(10)	Referenz auf die Organisation die der User zugeordnet ist
default_email_id	INT(10)	
status	INT(10)	Gibt den Status des Users an
name	TEXT	Name des Users
created	DATETIME	Erstelldatum des Users
updated	DATETIME	Änderungsdatum des Users

Tabelle 2.14.: tab:ost-user

## ost\_user\_account

Hier wird auf Basis der ost\_user Tabelle ein Account abgespeichert. Der Account beinhaltet eine Referenz auf einen User. Der Account besteht aus folgenden Informationen: der Status des Users, die Sprache des Users, in welcher Zeitzone dieser sich aufhält, den Usernamen, das Passwort und wann der User registriert wurde.

Column Name:	Datatype:	Content:
id	INT(11)	Eindeutige ID des Accounts. Dieser Wert ist rein technisch und hat neben der Eindeutigkeit keine weitere Aussagekraft.
user_id	INT(10)	Referenz auf den User dem der Account gehört
status	INT(11)	Den Status des Users
timezone	VARCHAR(64)	Gibt die Zeitzone an in dem sich der User befindet
lang	VARCHAR(16)	Die Sprache des Users
username	VARCHAR(64)	Username des Users
passwd	VARCHAR(128)	Das Passwort des Users in Hash form
backend	VARCHAR(32)	
extra	TEXT	Zusatz Informationen zum User
registered	TIMESTAMP	Hält fest wann sich der User registriert hat

Tabelle 2.15.: tab:ost-user-account

**ost\_ticket\_priotity**

Diese Tabelle enthält alle Informationen über die Priorität, die ein Ticket haben kann. Im gesamten kann einem Ticket eine von vier Prioritäten zugewiesen werden.

<b>Column Name:</b>	<b>Datatype:</b>	<b>Content:</b>
priority_id	TINYINT(4)	Eindeutige ID der Priorität. Dieser Wert ist rein technisch und hat neben der Eindeutigkeit keine weitere Aussagekraft.
priority	VARCHAR(60)	Beschreibt die Priorität
priority_desc	VARCHAR(30)	Leg fest wie die Prioritäten geordnet werden
priority_color	VARCHAR(7)	Leg die Farbe der Prioritäten fest
priority_urgency	TINYINT(1)	Leg fest welche Dringlichkeit die Priorität hat
ispublic	TINYINT(1)	Gib an ob die Priorität öffentlich ist oder nicht

Tabelle 2.16.: tab:ost-ticket-priotity

**ost\_help\_topic**

Diese Tabelle speichert die Hilfsthemas. Jedem Thema kann eine bestimmte Priorität zugeordnet werden. Zusätzlich können Themen auch bestimmten Abteilungen, Teams, Administrator\_innen oder Mitarbeiter\_innen zugeteilt werden.

<b>Column Name:</b>	<b>Datatype:</b>	<b>Content:</b>
topic_id	INT(11)	Eindeutige ID des Hilfsthemas. Dieser Wert ist rein technisch und hat neben der Eindeutigkeit keine weitere Aussagekraft.
topic_pid	INT(10)	Referenz zu einem Plugin für das Thema
isactive	TINYINT(1)	Leg fest ob das Thema aktiv ist oder nicht
ispublic	TINYINT(1)	leg fest ob das Thema für User zur Verfügung steht
noautoresp	TINYINT(3)	
flags	INT(10)	
status_id	INT(10)	Referenz zu dem Status
priority_id	TINYINT(4)	Referenz zu einem Plugin für das Thema

Tabelle 2.17.: tab:ost-help-topic



dept_id	INT(10)	Referenz zu einer Abteilung die dieses Thema bearbeiten
staff_id	INT(10)	Referenz zu einem Staff-Mitglied das diesem Thema bearbeitet
team_id	INT(10)	Referenz zu einem Team das diesem Thema bearbeitet
sla_id	INT(10)	Referenz zu einer SLA-Vorlage für das Thema
page_id	INT(10)	Referenz zu der Seite des Themas
sequence_id	INT(10)	Referenz zu der Sequenz für das Thema
sort	INT(10)	Gibt an wie das Thema gereiht werden soll
topic	VARCHAR(32)	Die Themen Beschreibung
number_format	VARCHAR(32)	
notes	TEXT	Zusätzliche Informationen zu dem Thema
created	DATETIME	Erstelldatum des Themas
updated	DATETIME	Änderungsdatum des Themas

Tabelle 2.18.: tab:ost-help-topic2

## 2.6.2. Beschreibung der Datenbankannotation von OSTicket

Die Datenbankannotation in diesem System ist in der Klasse `mysqli.php` verankert. In dieser Klasse sind Funktionen vorhanden, um eine Datenbankverbindung herzustellen, eine Query abzusetzen und die Verbindung wieder zu trennen. Um diese Funktionen nutzen zu können, muss die Klasse `mysqli.php` lediglich in dem zu bearbeiteten Quellcode durch ein Include Statement eingebunden werden. Die Klasse besitzt im gesamten eine globale Variable namens `$_db` sowie 32 Funktionen. Die Variable wird am Anfang der Klasse deklariert, nach dem Aufrufen der `db_connect`-Funktion hält sie ein Mysqli- Objekt. Im folgenden Abschnitt werden die wichtigsten Funktionen im Detail beschrieben.

### **function db\_connect**

Diese Funktion stellt die Verbindung zu einer Datenbank her. Um eine Verbindung herstellen zu können benötigt sie folgende Parameter:

- **\$host** ist die Hostadresse, unter der die Datenbank erreicht werden kann.
- **\$user** ist ein User, der die benötigten Rechte hat um lesend und schreibend auf die Datenbank zugreifen zu können.
- **\$passwd** ist das Passwort des Users.
- **\$options** ist ein Array, das Informationen über den Secure Sockets Layer (SSL) und den Namen der zu selektierenden Datenbank beinhaltet

Diese Funktion kann als Herzstück dieser Klasse bezeichnet werden, da immer zuerst eine Verbindung zu der Datenbank hergestellt werden muss, um mit ihr arbeiten zu können. Damit eine andere Funktion dieser Klasse genutzt werden können, muss zuerst die Funktion `db_connect` aufgerufen werden. Eine Verbindung besteht solange, bis Die Verbindung mit Hilfe der Funktion `db_close` beendet wird. Sollte in dieser Funktion ein Fehler vorliegen, so gibt sie den Wert `null` zurück.

Quelltext 2.5: mysqli.php/function-db\_connect1

```
21 function db_connect($host, $user, $passwd, $options =  
    array())  
22 {  
23     global $__db;  
24  
25     //Assert  
26     if(!strlen($user) || !strlen($host))  
27         return NULL;  
28  
29     if (!($__db = mysqli_init()))  
30         return NULL;
```

In diesem Abschnitt des Codes wird zuerst die Variable **\$\_\_db** als globale Variable deklariert. Anschließend wird durch die String-Funktion `strlen` geprüft, welche Länge die Parameter **\$user** und **\$host** haben. Hat einer der beiden Parameter eine Länge von 0, dann liefert die Funktion den Wert null.

Nach dem Überprüfen der beiden Parameter **\$user** und **\$host**, wird bei der nächsten IF-Anweisung geprüft, ob ein Mysqli- Objekt instantiiert werden kann. Ist das nicht der Fall, gibt die Funktion wieder den Wert null zurück,

Quelltext 2.6: mysqli.php/function-db\_connect2

```
32 if (isset($options['ssl']))
33     $__db->ssl_set(
34         $options['ssl']['key'],
35         $options['ssl']['cert'],
36         $options['ssl']['ca'],
37         null, null);
38 elseif(!$passwd)
39     return NULL;
```

In diesem Codeteil wird zuerst überprüft, ob im Array \$options Werte mit dem Schlüssel ssl abgelegt sind. Wenn dies der Fall ist, werden die Werte key, cert, und ca der Datenbankverbindung übergeben. Nach der Überprüfung des Arrays erfolgt die Kontrolle, ob ein Passwort gesetzt ist. Ist dies nicht der Fall, gibt die Funktion wieder null zurück.

Quelltext 2.7: mysqli.php/function-db\_connect3

```

41 $port = ini_get("mysqli.default_port");
42 $socket = ini_get("mysqli.default_socket");
43 $persistent = stripos($host, 'p:') === 0;
44 if ($persistent)
45     $host = substr($host, 2);
46 if (strpos($host, ':') !== false)
47 {
48     list($host, $portspec) = explode(':', $host);
49     // PHP may not honor the port number
50     // if connecting to 'localhost'
51     if ($portspec && is_numeric($portspec))
52     {
53         if (!strcasecmp($host, 'localhost'))
54             // XXX: Looks like PHP gethostbyname() is IPv4 only
55             $host = gethostbyname($host);
56         $port = (int) $portspec;
57     }
58     elseif ($portspec)
59     {
60         $socket = $portspec;
61     }
62 }
63
64 if ($persistent)
65     $host = 'p:' . $host;

```

In diesem Teilbereich des Codes werden Port, Socket und Hostname der Verbindung festgelegt. Zu Beginn des Abschnittes wird mit Hilfe der Funktion `ini_get` der default Wert vom MySQL Port und Socket den Variablen `$port` und `$socket` zugewiesen.

Anschließend wird mit Hilfe der Funktion `stripos` der Variable `$persistent` der Wert `true` zugewiesen, wenn der Value `'p:'` am Anfang des Strings `$host` steht oder `false` wenn `'p:'` nicht am Anfang des Strings zu finden ist.

Danach wird durch eine IF-Anweisung geprüft, welchen Wert die Variable \$persistent angenommen hat. Hat sie den Wert true, wird der Variable \$host durch die Funktion substr der vorherige Wert zugewiesen, lediglich die ersten zwei Zeichen des ursprünglichen Strings werden ausgelassen.

In der nächsten IF-Anweisung wird geprüft, ob sich im String der Variable \$host ein ':' befindet. Ist ein Doppelpunkt im String vorhanden, so wird alles, was sich Links vom Doppelpunkt befindet, der Variable \$host zugewiesen und alles, was rechts davon steht, wird der Variable \$portspec zugewiesen. Anschließend wird kontrolliert, ob die Variable \$portspec einen Wert hat und ob dieser Wert numerisch ist. Trifft beides zu, wird noch festgestellt, ob die Variable \$host den Wert 'localhost' beinhaltet. Hat die Variable \$host den Wert localhost nicht, wird der ihr durch die Funktion gethostbyname die IPv4 Adresse oder der unveränderte Hostname zugewiesen. Außerdem wird der Variable \$port der zu verwendende Port übergeben.

Quelltext 2.8: mysqli.php/function-db\_connect4

```
63 // Connect
64 $start = microtime(true);
65 if (!@$__db->real_connect($host, $user, $passwd, null,
        $port, $socket))
66 return NULL;
67 //Select the database, if any.
68 if(isset($options['db']))
        $__db->select_db($options['db']);
69
70 @$__db->query('SET NAMES "utf8"');
71 @$__db->query('SET CHARACTER SET "utf8"');
72 @$__db->query('SET COLLATION_CONNECTION=utf8_general_ci');
73 $__db->set_charset('utf8');
```

In diesem Abschnitt wird anhand der oben gesammelten Informationen die Verbindung zur Datenbank aufgebaut und jene ausgewählte Datenbank selektiert.

Anschließend werden Informationen über den verwendeten Zeichencode übergeben.

Der Rückgabewerte dieser Funktion ist, wenn keine Fehler vorkommt, eine Mysqli Objekt.

**function db\_query**

Mithilfe dieser Funktion lässt sich eine Query an die Datenbank schicken. Ist die Query ein SELECT Statement, bekommt man ein ResultSet zurück. Handelt es sich aber um eine Query mit einem INSERT-, UPDATE- oder DELETE- Statement, liefert die Funktion nach erfolgreichem Durchlaufen den Wert true. Ist dies nicht der Fall, liefert sie den Wert false. Um eine Query absetzen zu können, benötigt diese Funktion folgende Parameter:

- **\$query** Mit diesem Parameter wird die abzuschickende Query angegeben.
- **\$logError** Dieser Parameter ist standardmäßig immer true.
- **\$buffered** Dieser Parameter ist standardmäßig immer true.

Quelltext 2.9: mysqli.php/function-db\_query1

```
154 function db_query($query , $logError=true ,  
    $buffered=true)  
155 {  
156     global $ost , $__db;  
157  
158     if ($__db->unbuffered_result)  
159     {  
160         $__db->unbuffered_result->free();  
161         $__db->unbuffered_result = false;  
162     }
```

In diesem Teilabschnitt werden zunächst die Variablen \$ost und \$\_\_db als global-Variablen deklariert. Im Anschluss wird geprüft, ob die Variable \$\_\_db ein Unbuffered Result beinhaltet. Ist dies der Fall, wird das Unbuffered Result geleert und der Wert auf false gesetzt.

Quelltext 2.10: mysqli.php/function-db\_query2

```
163 $tries = 3;
164 do
165 {
166     $res = $__db->query($query ,
167         $buffered ? MYSQLI_STORE_RESULT : MYSQLI_USE_RESULT);
168     // Retry the query due to deadlock error (#1213)
169     // TODO: Consider retry on #1205
170     (lock wait timeout exceeded)
171     // TODO: Log warning
172 }
173 while (!$res && --$tries && $__db->errno == 1213);
```

Dieser Teil des Codes schickt mit Hilfe der query Funktion eine Anfrage mit der Query an die Datenbank und speichert das Ergebnis in der Variable \$res. Anschließend wird kontrolliert, ob die Abfrage erfolgreich war und ob \$\_\_db->errno gleich 1213 ist. Zusätzlich wird die Variable \$tries um eins verkleinert. Treffen alle Argumente zu, war das Abschicken der Query erfolglos und wird noch einmal vorgenommen. Nach drei Versuchen bricht die do-while-Schleife ab und es wird nicht mehr versucht eine Anfrage an die Datenbank zu schicken. War das Abschicken der Query hingegen erfolgreich, wird die do-while-Schleife ebenfalls abgebrochen und die nächste IF-Anweisung wird nicht durchgeführt.



Quelltext 2.11: mysqli.php/function-db\_query3

```

173 if(!$res && $logError && $ost)
174 {
175     //error reporting
176     // Allow $logError()
177     // callback to determine if logging is necessary
178     if (is_callable($logError) &&
179         !($logError($__db->errno)))
180         return $res;
181
182     $msg='['.$query.']'. "\n\n".db_error();
183     $ost->logDBError('DB Error #'.db_errno(), $msg);
184     //echo $msg; #uncomment during debugging or dev.
185 }
186
187 if (is_object($res) && !$buffered)
188     $__db->unbuffered_result = $res;
189
190 return $res;

```

Wenn, wie im oberen Abschnitt schon erwähnt, die do-while-Schleife den Versuch eine Query an die Datenbank zu schicken abgebrochen hat, wird in einem nächsten Schritt geprüft ob, die Variable \$res etwas beinhaltet und ob die Variablen \$logError und \$ost deklariert wurden. Treffen alle Parameter zu, wird bei der nächsten IF-Anweisung kontrolliert, ob die Variable \$logError einen Wert beinhaltet, der als Funktion aufgerufen werden kann und ob der Aufruf dieser Funktion mit dem Parameter \$\_\_db->errno den Wert false zurück gibt. Trifft dies auch zu, wird die Variable \$res mit dem Wert null zurück gegeben. Anschließend wird der Variable \$msg der Fehlertext übergeben und mit der Variable \$ost wird die Funktion logDBError aufgerufen. Als Parameter werden dieser Funktion der Errorcode und der Errortext mitgegeben. In der letzten IF-Anweisung soll noch geprüft werden, ob die Variable \$res ein Objekt ist und ob die Variable \$buffered den Wert false hält. Ist dies der Fall, wird der Wert der Variable \$res dem Datenfeld unbuffered\_result des Objektes \$\_\_db übergeben. Zuletzt wird noch die Variable \$res zurückgegeben.

### 2.6.3. Tabellenbeschreibung der Datenbank von unserem Java EE Prototyp

Dieser Teil der Dokumentation beschreibt die Datenbank von unserem Java EE Prototypen. Dieser erfüllt grundsätzlich alle Anforderungen, die an das Ticketsystem gestellt wurden.

Hinter der Anwendung steht die hier beschriebene Datenbank. Sie wurde konzipiert, möglichst einfach und übersichtlich zu sein. Es wurde auch darauf geachtet, dass keine unnötigen Tabellen verwendet werden müssen, da wir anhand dieser Anwendung zeigen möchten, wie viele überflüssige Funktionen dem OSTicketsystem zugrunde liegen. Für eine grobe Übersicht, finden Sie hier ein EER-Diagramm der Datenbank.

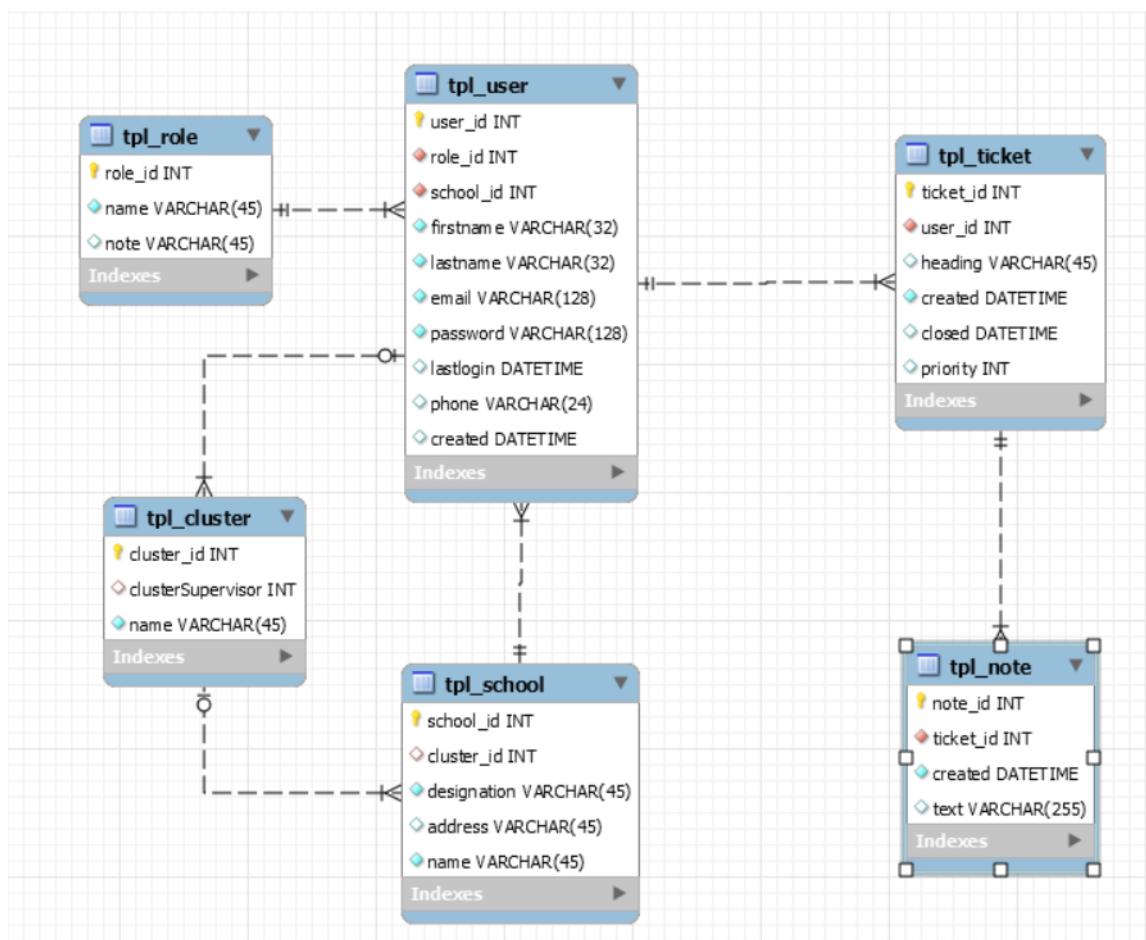


Abbildung 2.16.: EER-Diagramm-Java-EE-Anwendung

## **tpl\_user**

In dieser Tabelle werden die User des Ticketsystems gehalten. Von jedem User werden der Vorname, Nachname, E-Mail-Adresse und das Passwort gespeichert. Optional kann auch der letzte Login, die Telefonnummer und das Datum, an dem der Account erstellt wurde, gespeichert werden. Ein User muss einer Schule zugewiesen sein, um vollständig im System registriert zu werden. Ein User kann optional auch die Funktion des Supervisors eines Clusters einnehmen. **WICHTIG:** lastlogin und created werden nach folgender Form in der Datenbank gespeichert "YYYY-MM-DD HH:MM:SS".

<b>Column Name:</b>	<b>Datatype:</b>	<b>Content:</b>
user_id	INT	Eindeutige ID des User. Dieser Wert ist rein technisch und hat neben der Eindeutigkeit keine weitere Aussagekraft.
role_id	INT(	Referenz zu der Rolle die der User besitzt
school_id	INT	Referenz zu der Schule vom User
firstname	VARCHAR(32)	Vorname des Users
lastname	VARCHAR(32)	Nachname des Users
email	VARCHAR(128)	E-Mail-Adresse des Users
password	VARCHAR(128)	Das Passwort des Users in Hash Form
lastlogin	DATETIME	Das Datum, in oben angegebener Form, an dem sich der User zuletzt angemeldet hat
password	VARCHAR(24)	Das Passwort des Users in Hash Form
created	DATETIME	Das Datum, in oben angegebener Form, an dem der User angelegt worden ist

Tabelle 2.19.: tab:tpl-user

## **tpl\_ticket**

In dieser Tabelle werden die erstellten Tickets des Ticketsystems gespeichert. Das Ticket wird genau einem User zugeordnet und kann mehrere Notizen (notes) beinhalten. Des weiteren besteht ein Ticket aus einem Header, der Priorität und an welchem Datum das Ticket erstellt und geschlossen wurde. WICHTIG: closed und created werden nach folgender Form in der Datenbank gespeichert "YYYY-MM-DD HH:MM:SS".

<b>Column Name:</b>	<b>Datatype:</b>	<b>Content:</b>
ticket_id	INT	Eindeutige ID des Tickets. Dieser Wert ist rein technisch und hat neben der Eindeutigkeit keine weitere Aussagekraft.
user_id	INT	Referenz zum User zu dem das Ticket gehört
heading	VARCHAR(45)	Die Überschrift des Tickets
created	DATETIME	Das Datum, in oben angegebener Form, an dem das Ticket angelegt worden ist
closed	DATETIME	Das Datum, in oben angegebener Form, an dem das Ticket geschlossen worden ist
priority	INT	Hier wird die Priorität des Tickets angegeben. Kann einen Wert von 0-4 haben, der default Wert ist 0.

Tabelle 2.20.: tab:tpl-ticket

## **tpl\_note**

In dieser Tabelle werden die Notizen (notes) für die Tickets gespeichert. Jeder Notiz wird ein Ticket zugewiesen. Ein Ticket kann mehrere Notizen enthalten, umgekehrt aber gehört eine Notiz immer genau zu einem Ticket. Eine Notiz besteht aus einem Text und einem Datum, das angibt, wann die Notiz erzeugt worden ist. **WICHTIG:** created wird in folgendem Format in der Datenbank gespeichert: "YYYY-MM-DD HH:MM:SS".

<b>Column Name:</b>	<b>Datatype:</b>	<b>Content:</b>
note_id	INT	Eindeutige ID der Notiz. Dieser Wert ist rein technisch und hat neben der Eindeutigkeit keine weitere Aussagekraft.
ticket_id	INT	Referenz zum Ticket dem diese Notiz angehört
created	DATETIME	Das Datum, in oben angegebener Form, an dem die Notiz angelegt worden ist
text	VARCHAR(255)	Ein Text der Informationen zum Ticket beinhaltet

Tabelle 2.21.: tab:tpl-note

### **tpl\_role**

In dieser Tabelle werden die Rollen gespeichert, die ein User annehmen kann. Eine Rolle besteht lediglich aus einem Namen und einer Notiz. Ein User kann genau eine Rolle haben, aber eine Rolle kann umgekehrt von mehreren Usern genutzt werden.

Column Name:	Datatype:	Content:
role_id	INT	Eindeutige ID der Rolle. Dieser Wert ist rein technisch und hat neben der Eindeutigkeit keine weitere Aussagekraft.
name	VARCHAR(45)	Name der Rolle
note	VARCHAR(45)	Informationen über die Rolle

Tabelle 2.22.: tab:tpl-role

### **tpl\_school**

In dieser Tabelle werden die Schulen gespeichert, die dieses Ticketsystem verwenden. Eine Schule wird mit ihrer Bezeichnung, Adresse und Namen abgespeichert. Des Weiteren kann eine Schule einem Cluster zugeordnet werden.

Column Name:	Datatype:	Content:
school_id	INT	Eindeutige ID der Schule. Dieser Wert ist rein technisch und hat neben der Eindeutigkeit keine weitere Aussagekraft.
cluster_id	INT	Referenz zu dem Cluster dem die Schule angehört
designation	VARCHAR(45)	Schulbezeichnung der Schule
address	VARCHAR(45)	Die Adresse der Schule
name	VARCHAR(45)	Name der Schule

Tabelle 2.23.: tab:tpl-school

**tpl\_cluster**

In dieser Tabelle werden die Cluster festgelegt. Als Cluster wird eine Mengen von Schulen, die zu einer Organisation zusammen geschlossen wurden, bezeichnet. Jeder Cluster hat einen Supervisor, der für die Bearbeitung der anfallenden Tickets in ebendiesem Cluster verantwortlich ist. Der Supervisor wird per Referenz zur User-Tabelle festgelegt. Ein Cluster hat immer genau einen Supervisor. Zusätzlich wird noch ein Name für den Cluster festgelegt.

<b>Column Name:</b>	<b>Datatype:</b>	<b>Content:</b>
cluster_id	INT	Eindeutige ID des Clusters. Dieser Wert ist rein technisch und hat neben der Eindeutigkeit keine weitere Aussagekraft.
clusterSupervisor	INT	Referenz zu dem User der in diesem Cluster der Supervisor ist
name	VARCHAR(45)	Name des Clusters

Tabelle 2.24.: tab:tpl-cluster

## 2.6.4. Beschreibung der Datenbankanbindung von unserem Java EE Prototyp

Da unsere Anwendung nur ein Prototyp ist, wurde auch die Datenbankanbindung nur mit den wichtigsten Funktionen konzipiert. Deshalb ist die Klasse Datenbankanbindung.java auch sehr statisch und redundant geschrieben. Die Datenbankanbindung wurde mit Hilfe der Java Api JDBC realisiert.

Mit Hilfe der get-Methoden lassen sich die einzelnen Tabellen aus der Datenbank auslesen und durch die insert-Methoden werden Einträge in die Tabelle eingefügt. Mit der Methode dbconnect kann die Verbindung zu der Datenbank hergestellt werden.

Am Anfang der Klasse werden drei String Variablen mit Informationen über den Datenbank-Zugang initialisiert.

- **\$url** Die URL unter der die Datenbank zu erreichen ist. In diesem Fall liegt die Datenbank Local auf einem Rechner, da es sich noch um einen Prototypen handelt.
- **\$username** Ist ein User, der die benötigten Rechte hat, um lesend und schreibend auf die Datenbank zugreifen zu können. In unserem Fall greifen wir mit dem root User auf die Datenbank zu, da sie nur Local gespeichert ist und wir uns noch keine Gedanken über Sicherheit machen müssen.
- **\$password** Ist das Passwort des User.

Quelltext 2.12: Datenbankanbindung.java/Datenfelder

```
30 private String url =  
    "jdbc:mysql://localhost:3306/java_ee_database";  
31 private String username = "root";  
32 private String password = "";
```



Im folgenden werden die Methoden Klasse Datenbankanbindung.java im Detail beschrieben.

### Methode dbconnect

Diese Methode stellt eine Verbindung zu der Datenbank her und liefert ein Statement Objekt zurück. Das Statement Objekt wird benötigt, um eine Query an die Datenbank zu schicken. Um ein Statement Objekt erzeugen zu können, muss zuerst ein Connection Objekt erzeugt werden. Dies erfolgt durch die Methode getConnection mit den Parameter url, username, password. Konnte keine Verbindung zu der Datenbank hergestellt werden, wird durch den Catch-Block die SQLException gefangen und der Wert null zurück gegeben.

Quelltext 2.13: Datenbankanbindung.java/Methode-dbconnect

```
44 public Statement dbconnect()
45 {
46     try
47     {
48         Connection connection = (Connection)
49         DriverManager.getConnection(url, username, password);
50         Statement stmt = connection.createStatement();
51         return stmt;
52     } catch(SQLException e)
53     {
54         return null;
55     }
56 }
```

**Methode insertRole**

Bei dieser Methode wird ein Insert-Statement erzeugt und anschließend an die Datenbank geschickt, zurück bekommen wir den String 'Das Insert hat funktioniert.' oder null, wenn das Insert nicht funktioniert hat. Als Parameter werden die Values benötigt, welche man in die Datenbank schreiben möchte. Zuerst wird mit der Methode dbconnect ein Statment Objekt erstellt und darauffolgend wird mit Hilfe der executeUpdate Methode die Query an die Datenbank geschickt. Ist das Abschicken der Query nicht erfolgreich, bekommt man den Wert null zurück.

Dieser Methodenaufbau finden wir bei allen insert-Methoden dieser Klasse.

Quelltext 2.14: Datenbankanbindung.java/Methode-dbconnect

```
271 public String insertRole(String name, String note)
272 {
273     try
274     {
275         dbconnect().executeUpdate("INSERT INTO tpl_role
276             (name, note) VALUES (" + name + "\", \"" +
277             note + "\"");
278         return "Das Insert hat funktioniert";
279     }
280     catch(SQLException e)
281     {
282         return null;
283     }
```

## **Methode getRoles**

Mit Hilfe der Methode getRoles lassen sich alle Rollen aus der Datenbanktabelle role auslesen. Zuerst wird in dieser Methode eine ArrayList roles initialisiert, anschließend werden String Variablen definiert, die benötigt werden, um aus dem ResultSet die Datenfelder der Role zu lesen. Im try-Block wird nun ein SELECT-Statement an die Datenbank geschickt und ein ResultSet Objekt rs erzeugt. Um aus dem ResultSet die einzelnen Rollen auszulesen, benötigen wir eine while-Schleife. Diese iterieren solange durch das ResultSet, bis kein Eintrag mehr vorhanden ist (bis rs.next() false zurück gibt). In dieser while-Schleife wird zuerst ein role Objekt erzeugt und mit Hilfe der set-Methoden die Datenfelder dieses Rollenobjekts gesetzt. Zum Schluss wird das Rollenobjekt noch der ArrayList roles hinzugefügt. Wurde durch alle Einträge des ResultSet iteriert, wird das Statement geschlossen. Tritt während dem Durchlaufen des Try-Blockes ein Fehler auf, so wird der Wert null durch den Catch-Block zurückgegeben.

Diesen Methodenaufbau finden wir bei allen get-Methoden dieser Klasse.

Quelltext 2.15: Datenbankbindung.java/Methode-getRoles

```
59 public ArrayList getRoles()
60 {
61     ArrayList <tpl_role> roles = new ArrayList<>();
62
63     String value = "role_id";
64     String value1 = "name";
65     String value2 = "note";
66     try
67     {
68         ResultSet rs = dbconnect().executeQuery("SELECT *
69         FROM tpl_role");
70
71         while(rs.next())
72         {
73             tpl_role role = new tpl_role();
74             role.setName(rs.getString(value1));
75             role.setId(rs.getInt(value));
76             role.setNote(rs.getString(value2));
77             roles.add(role);
78         }
79         rs.close();
80         return roles;
81     } catch (SQLException ex)
82     {
83         return null;
84     }
85 }
```

# 3. Problemanalyse

## 3.1. USE-Case-Analyse

Akteure:

- Systembetreuer
- Anwender (IT-Manager und eventuell Lehrer, in Folgendem Anwender genannt)

<b>Name</b>	Anmelden am Portal
<b>ID</b>	C00
<b>Beschreibung</b>	Ein Anwender/Systembetreuer nutzt seine Login-Daten um sich in das System einzuwählen.
<b>Akteure</b>	Anwender, Systembetreuer
<b>Häufigkeit</b>	5/5
<b>Auslöser</b>	Der Anwender möchte das Portal benutzen.
<b>Bedingungen</b>	Anmeldedaten vorhanden und Rechte vergeben.
<b>Endzustand</b>	Eine Session wurde geöffnet und der Anwender kann das System nutzen.
<b>Hauptablauf</b>	1. Benutzerdaten eingeben 2. Login Vorgang initiieren
<b>Ausnahmen</b>	Error → Anmeldedaten prüfen. Error → Anmeldedaten neu beziehen.

Tabelle 3.1.: Use-Case C00

<b>Name</b>	Erstellung eines Standardtickets
<b>ID</b>	C01
<b>Beschreibung</b>	Ein Anwender meldet sich am Portal an und möchte ein neues Netzwerkgerät bestellen. Dieser eröffnet ein neues Standardticket und beschreibt den Grund für die Anschaffung. Der Anwender reicht das Ticket ein.
<b>Akteure</b>	Anwender
<b>Häufigkeit</b>	2/5
<b>Auslöser</b>	Es wird eine Komponente/Ressource für die Schule benötigt.
<b>Bedingungen</b>	C00
<b>Endzustand</b>	Ticket wurde in der Datenbank gespeichert.
<b>Hauptablauf</b>	<ol style="list-style-type: none"><li>1. Formular öffnen</li><li>2. Formular ausfüllen</li><li>3. Ticket überprüfen</li><li>4. Ticket einreichen</li></ol>
<b>Ausnahmen</b>	Error - Ticket kann nicht abgegeben werden → C01 erneut ausführen.

Tabelle 3.2.: Use-Case C01

<b>Name</b>	Erstellung eines Incident
<b>ID</b>	C02
<b>Beschreibung</b>	Ein Anwender meldet sich am Portal an und öffnet ein Ticket, er meldet ein Problem, das er selbst nicht lösen kann. Der Anwender markiert das Ticket als Incident und speichert es.
<b>Akteure</b>	Anwender
<b>Häufigkeit</b>	4/5
<b>Auslöser</b>	Ein für den Anwender nicht lösbares technisches Problem.
<b>Bedingungen</b>	C00, Auslöser
<b>Endzustand</b>	Incident wurde in der Datenbank gespeichert und der Systembetreuer hat eine Benachrichtigung erhalten.
<b>Hauptablauf</b>	<ol style="list-style-type: none"><li>1. Formular öffnen</li><li>2. Formular ausfüllen</li><li>3. Incident überprüfen</li><li>4. Incident einreichen</li></ol>
<b>Ausnahmen</b>	Error - Ticket kann nicht abgegeben werden → C02 erneut ausführen.

Tabelle 3.3.: Use-Case C02

<b>Name</b>	Ticketstatus prüfen
<b>ID</b>	C03
<b>Beschreibung</b>	Ein Anwender meldet sich am Portal an und sieht sich seine eigenen Tickets an. Er kann einsehen ob der Systembetreuer das Ticket erhalten hat und wie weit seine Bestellung vorgeschritten ist.
<b>Akteure</b>	Anwender
<b>Häufigkeit</b>	3/5
<b>Auslöser</b>	Anwender hat Anfrage erhalten
<b>Bedingungen</b>	C00, zu überprüfendes Ticket muss vorhanden sein
<b>Endzustand</b>	Keine Veränderungen am System
<b>Hauptablauf</b>	<ol style="list-style-type: none"> <li>1. Ticket öffnen</li> <li>2. Status einsehen</li> </ol>
<b>Ausnahmen</b>	Error – Ticket kann nicht überprüft werden → C03 erneut ausführen.

Tabelle 3.4.: Use-Case C03

<b>Name</b>	Ticketlöschung beantragen
<b>ID</b>	C04
<b>Beschreibung</b>	Ein Anwender meldet sich am Portal an und beantragt die Löschung seines Tickets.
<b>Akteure</b>	Anwender
<b>Häufigkeit</b>	1/5
<b>Auslöser</b>	Problem hat sich erübrigt; Ticket ist falsch
<b>Bedingungen</b>	C00, zu löschendes Ticket muss vorhanden sein
<b>Endzustand</b>	Systembetreuer hat die Anfrage auf Löschung erhalten
<b>Hauptablauf</b>	<ol style="list-style-type: none"> <li>1. Ticket öffnen</li> <li>2. Löschung beantragen</li> </ol>
<b>Ausnahmen</b>	Error – Löschvorgang nicht erfolgreich → C04 erneut ausführen.

Tabelle 3.5.: Use-Case C04



<b>Name</b>	Ticket löschen
<b>ID</b>	C05
<b>Beschreibung</b>	Der Systembetreuer meldet sich am Portal an und kümmert sich um Löschanfragen
<b>Akteure</b>	Systembetreuer
<b>Häufigkeit</b>	1/5
<b>Auslöser</b>	C04
<b>Bedingungen</b>	C00, zu löschendes Ticket muss vorhanden sein
<b>Endzustand</b>	Das Ticket wurde entfernt
<b>Hauptablauf</b>	<ol style="list-style-type: none"> <li>1. Ticket öffnen</li> <li>2. Ticket löschen</li> </ol>
<b>Ausnahmen</b>	Error – Löschvorgang nicht erfolgreich → C05 erneut ausführen.

Tabelle 3.6.: Use-Case C05

<b>Name</b>	Ticketstatus ändern
<b>ID</b>	C06
<b>Beschreibung</b>	Der Systembetreuer meldet sich am Portal an und sieht die Tickets der Anwender in seinem Cluster. Er ändert den Status eines Tickets in Bearbeitung und kümmert sich um die Anfrage.
<b>Akteure</b>	Systembetreuer
<b>Häufigkeit</b>	4/5
<b>Auslöser</b>	C01
<b>Bedingungen</b>	C00, Ticket vorhanden
<b>Endzustand</b>	Der Ticketstatus wurde geändert
<b>Hauptablauf</b>	<ol style="list-style-type: none"> <li>1. Ticket öffnen</li> <li>2. Ticketstatus ändern</li> </ol>
<b>Ausnahmen</b>	Error – Statusänderung nicht erfolgreich → C06 erneut ausführen.

Tabelle 3.7.: Use-Case C06

<b>Name</b>	Incident bearbeiten
<b>ID</b>	C07
<b>Beschreibung</b>	Der Systembetreuer erhält eine E-Mail-Benachrichtigung über einen Incident. Er klickt auf den mitgelieferten Link und meldet sich am Portal an. Er bearbeitet den Incident mit erhöhter Priorität.
<b>Akteure</b>	Systembetreuer
<b>Häufigkeit</b>	3/5
<b>Auslöser</b>	C02
<b>Bedingungen</b>	C00, Incident vorhanden
<b>Endzustand</b>	Incident ist abgearbeitet
<b>Hauptablauf</b>	<ol style="list-style-type: none"> <li>1. Benachrichtigung erhalten</li> <li>2. Incident öffnen</li> <li>3. Incident bearbeiten</li> <li>4. C08</li> </ol>
<b>Ausnahmen</b>	Error – Incident öffnen nicht erfolgreich → C07 erneut ausführen.

Tabelle 3.8.: Use-Case C07

### 3.1.1. Ablaufbeschreibung

#### C00:

Der Benutzer wird aufgefordert seine Benutzerdaten einzugeben. Hat er die Daten richtig eingegeben wird er angemeldet. Sind die Daten falsch kommt er zurück zur Dateneingabe.

#### C01:

Der Benutzer muss zuerst ein Formular öffnen, um ein Ticket erstellen zu können. Anschließend muss das Formular ausgefüllt werden.

Nach dem Überprüfen des Tickets kann der Benutzer sich entscheiden ob er das Ticket abschickt oder ob er Änderungen vornehmen möchte.

Wenn das Abschicken fehlgeschlagen ist kommt er zum Anfang zurück. Wenn das abschicken erfolgreich war, wird das Ticket in der Datenbank gespeichert.

#### C02:

Der Benutzer muss zuerst ein Formular öffnen um einen Incident erstellen zu können. Anschließend muss das Formular ausgefüllt werden. Nach dem Überprüfen des Incident kann der Benutzer sich entscheiden, ob er den Incident abschickt oder ob er Änderungen vornehmen möchte.

Wenn das Abschicken fehlgeschlagen ist kommt er zum Anfang zurück. Wenn das

<b>Name</b>	Ticket schließen
<b>ID</b>	C08
<b>Beschreibung</b>	Der Systembetreuer meldet sich am Portal an und schließt ein Ticket bzw. einen Incident nach dessen Erledigung.
<b>Akteure</b>	Systembetreuer
<b>Häufigkeit</b>	5/5
<b>Auslöser</b>	C01, C02
<b>Bedingungen</b>	C00, Ticket vorhanden und abgearbeitet
<b>Endzustand</b>	Ticket ist geschlossen
<b>Hauptablauf</b>	1. Ticket öffnen 2. Ticket schließen
<b>Ausnahmen</b>	Error – Ticket konnte nicht geschlossen werden → C08 erneut ausführen.

Tabelle 3.9.: Use-Case C08

abschicken erfolgreich war wird der Incident in der Datenbank gespeichert und der Systembetreuer erhält eine Benachrichtigung.

**C03:**

Der Benutzer muss das Formular öffnen um den Status zu sehen. Ist das Öffnen fehlgeschlagen kommt er wieder zum Ausgangspunkt und kann es nochmal versuchen.

**C04:**

Um die Löschung beantragen zu können muss der Benutzer zuerst das Ticket öffnen. Danach kann er die Löschung beantragen. Schlägt dies fehl kommt er wieder zurück an den Anfang und kann es nochmal versuchen. War der Antrag auf Löschung erfolgreich erhält der Systembetreuer eine Anfrage zur Löschung.

**C05:**

Um ein Ticket zu löschen muss der Systembetreuer das Ticket öffnen und löschen. Schlug dies fehl kommt er wieder zurück an den Anfang und kann es nochmal versuchen. War das Löschen erfolgreich wurde das Ticket aus der Datenbank entfernt.

**C06:**

Um den Ticketstatus ändern zu können muss der Systembetreuer das Ticket öffnen und ändern. Schlug dies fehl kommt er wieder zurück an den Anfang und kann es nochmal versuchen. War das ändern erfolgreich wurde der Ticketstatus geändert.

**C07:**

Um ein Ticket schließen zu können muss der Systembetreuer das Ticket öffnen um es dann zu schließen. Schlug dies fehl kommt er wieder zurück an den Anfang und kann es nochmal versuchen. War das schließen erfolgreich ist das Ticket geschlossen.

## 3.2. User-Interface-Design

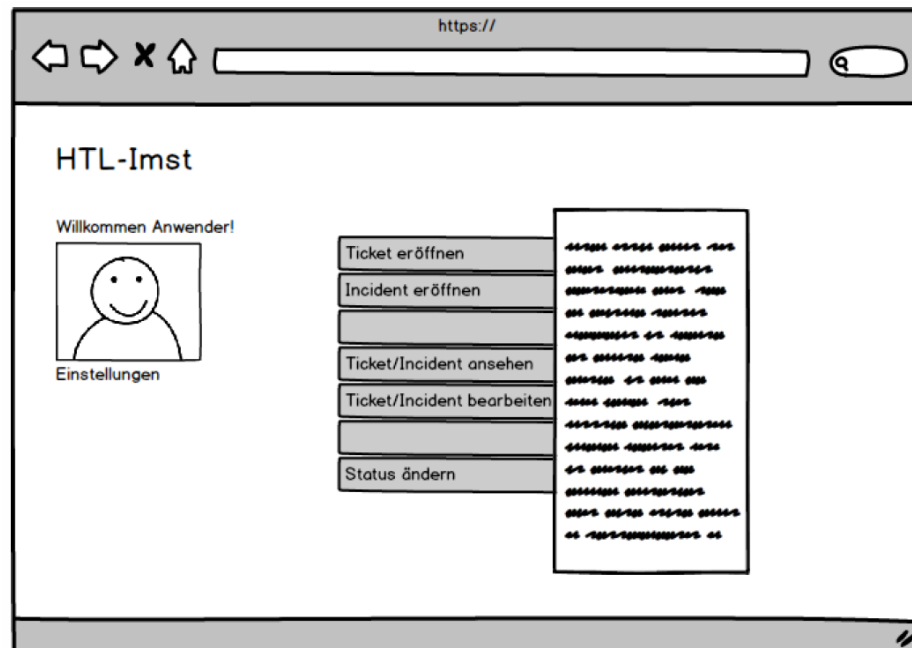


Abbildung 3.1.: Mockup Anwendersicht

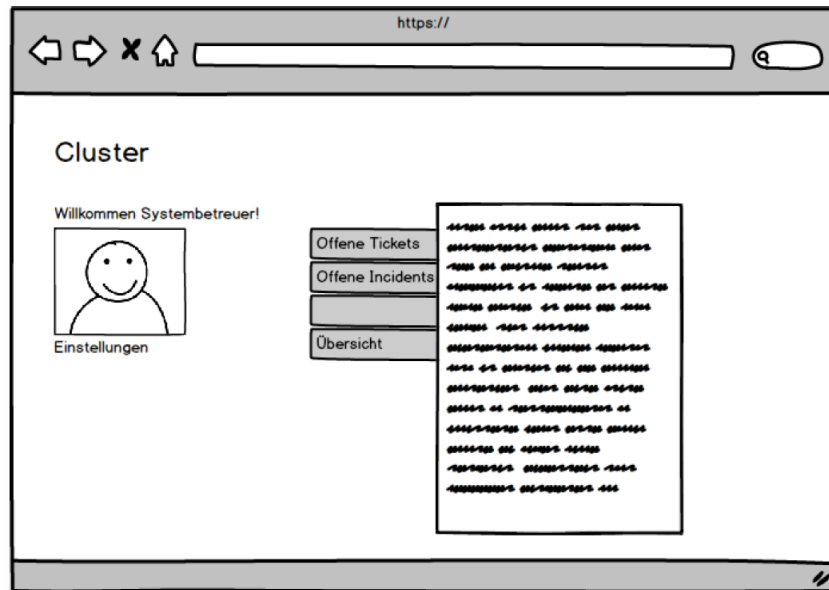


Abbildung 3.2.: Mockup Systembetreuer

### 3.3. Prototyp

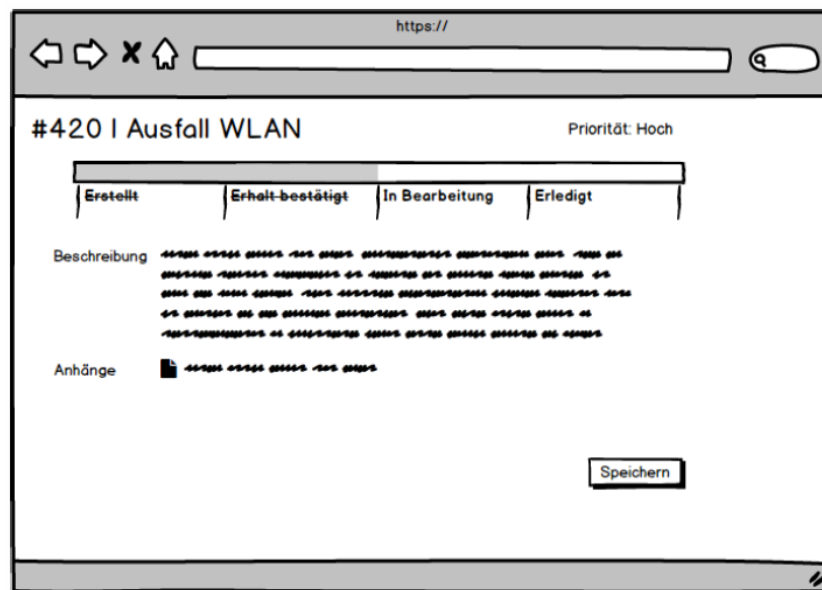


Abbildung 3.3.: Mockup Ticketstatus

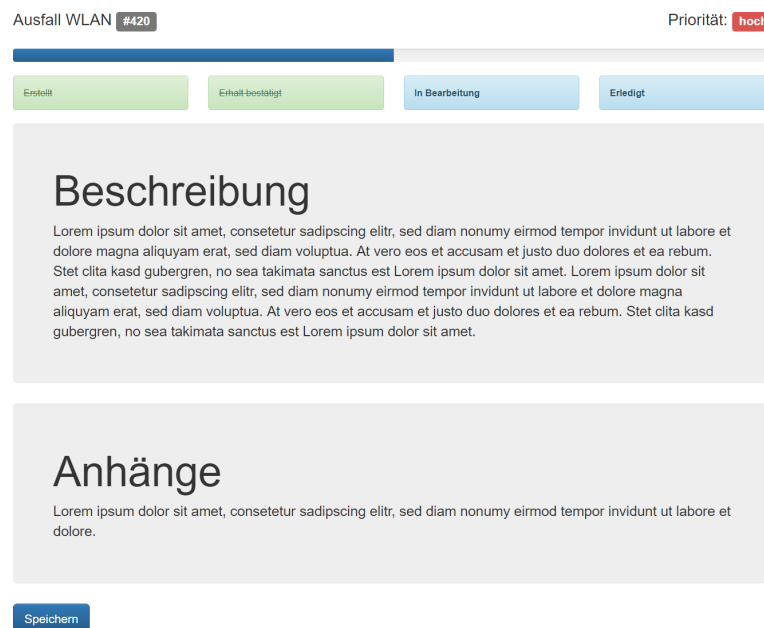


Abbildung 3.4.: Prototyp Ticketstatus

## 4. Entwurf/Machbarkeitsfeststellung eines kleinen Java EE Ticketsystems

Aufgrund der bereits geschilderten Nicht Durchführbarkeit einer Adaption von OSTicket wurde ein Prototyp für ein Ticketsystem entworfen, welcher die vom Landesschulrat und den Tiroler Schulen benötigten Funktionen mitbringt. Die Anforderungen für dieses System sind im Grunde die gleichen, welche zu Beginn an die OSTicket-Adaption gestellt wurden.

Ein Auszug dieser Anforderungen sind:

- Usability: Das System soll ohne lange Schulungsphasen oder Einlernzeit verwendet werden können. Unter Verwendung ist hauptsächlich das Absetzen von Supporttickets definiert.
- Mobility: Das Web-Frontend soll auch auf Smartphones und Tablets verwendet werden können und die gleichen Funktionen wie auf dem PC liefern.
- Adaptability: Sollten sich Anforderungen, Best Practices oder Sicherheitsanforderungen ändern, sollen diese mit so geringem Aufwand als möglich implementiert werden können.

### 4.1. Technologie

Für den Prototyp des Ticketsystems wurde Java Enterprise Edition ausgesucht. Diese Entscheidung basiert auf der Absicht, die bei OSTicket gezogenen Schlüsse zu be-



achten und die Probleme die bei OSTicket auftraten zu vermeiden. JavaEE erscheint hierfür besonders geeignet, da es (in dem Verwendungsmodus, der an der Schule gelehrt wurde) von sich aus das MVC-Entwurfsmuster anwendet (mehr im nächsten Abschnitt). Des Weiteren eignet sich JavaEE für die Beseitigung der Schwächen OSTickets durch die relativ strengen Sprachkonventionen und die (beinahe) unausweichliche Objektorientierung.

## **4.2. Architektur**

Als Basis für die Systemarchitektur wird das Model View Controller Muster verwendet. Das bedeutet die Trennung zwischen JavaBeans (Model), die direkt mit der Persistenzebene (Datenbank) arbeitet, der Benutzerschnittstelle (View; Webschicht) und der Logik (Controller; Anwendungsschicht).

Das Lehrbuch fasst das Entwurfsmuster wie folgt zusammen und bringt dessen Sinn sowie Existenzberechtigung im Evaluationsprogramm „Ticketsystem“ auf den Punkt:

Das MVC ist ein Muster, das vorgibt, wie Darstellung, Logik und Daten in einer Applikation getrennt werden sollen. Ziel dieser Trennung ist die Verbesserung der Programmstruktur und damit die Wartbarkeit, Erweiterbarkeit, und Wiederverwendbarkeit des Codes. Das Modell kapselt die Daten und enthält je nach MVC-Ausprägung ggf. auch die fachliche Logik. Die View visualisiert das Modell und der Controller realisiert die Anwendungssteuerung. Der Controller reagiert auf Benutzerinteraktionen innerhalb der View und aktualisiert ggf. die Daten am Modell. Die View wiederum passt sich je nach Ausprägung des MVC entweder automatisch an das veränderte Modell an oder wird durch den Controller über die Ausprägung informiert. Schiesser und Schmollinger (2015)

## 4.3. Benutzerschnittstellen

Benutzerschnittstellen werden in Java Enterprise Edition - Anwendungen mit der Technologie „Java Server Faces“ umgesetzt. Früher wurde „Java Server Pages“ verwendet, welches 1999 erschien und mittlerweile veraltet ist. Java Server Faces ist ein Framework für Oberflächen von Webanwendungen. Folgende Teilkomponenten werden grundsätzlich für JSF benötigt:

- Java Development Kit wurde installiert und eingebunden
- Servlet-Container (bspw. Apache Tomcat) ist einsatzbereit

Im unteren Listing findet sich beispielhaft ein Auszug aus der XHTML-Datei List.xhtml. Sie oberflächlich gesehen dafür zuständig, eine Tabelle von Tickets und deren detaillierten Eigenschaften anzuzeigen. Des Weiteren finden sich auch Navigationselemente in der Datei. Solche Tags tragen die Bezeichnung „commandLink“. Definiert in ihrer „action“-Eigenschaft befindet sich ein Expression-Language (EL) Ausdruck welcher auf eine Methode der TicketController Klasse verweist. Ausdrücke der EL sind immer umschlossen von geschweiften Klammern und geführt von einer Raute: `{Klassenname.Datenfeld}`, als kleines Beispiel.

Eine weitere Komponente die hervorzuheben ist, ist der DataTable. Diese Komponente stemmt die Hauptaufgabe der Seite. Sie bezieht eine Liste von Tickets mit dem EL-Ausdruck „`{ticketController.items}`“; eine Laufvariable `var=item` hält also ein Ticket. Die Laufvariable durchläuft jedes Ticket genau ein mal, vergleichbar mit einer klassischen For-Schleife. So werden die Eigenschaften jedes Tickets in einer Zeile des DataTable dargestellt.

Quelltext 4.1: List.xhtml

```
0 <h:panelGroup rendered="#  
1 {  
2     tplTicketController.items.rowCount > 0  
3 }  
4 ">
```

```
5 <h:outputText value="#"
6 {
7     tplTicketController.pagination.pageFirstItem + 1
8 }
9 ..#
10 {
11     tplTicketController.pagination.pageLastItem + 1
12 }
13 /#
14 {
15     tplTicketController.pagination.itemsCount
16 }
17 ">&nbsp;
18 <h:commandLink action="#"
19 {
20     tplTicketController.previous
21 }
22 " value="#"
23 {
24     bundle.Previous
25 }
26 #
27 {
28     tplTicketController.pagination.pageSize
29 }
30 " rendered="#"
31 {
32     tplTicketController.pagination.hasPreviousPage
33 }
34 ">&nbsp;
35 <h:commandLink action="#"
36 {
37     tplTicketController.next
38 }
39 " value="#"
```

```

40 {
41     bundle.Next
42 }
43 #
44 {
45     tplTicketController.pagination.pageSize
46 }
47 " rendered="#"
48 {
49     tplTicketController.pagination.hasNextPage
50 }
51 "/>&nbsp;
52 <h:dataTable value="#"
53 {
54     tplTicketController.items
55 }
56 " var="item" border="0" cellpadding="2" cellspacing="0"
    rowClasses="jsfcrud_odd_row,jsfcrud_even_row"
    rules="all" style="border:solid 1px">
57 <h:column>
58 <f:facet name="header">
59 <h:outputText value="#"
60 {
61     bundle.ListTplTicketTitle_ticketId
62 }
63 "/>
64 </f:facet>
65 <h:outputText value="#"
66 {
67     item.ticketId
68 }
69 "/>
70 </h:column>
71 </h:dataTable>
72 </h:panelGroup>

```

## 4.4. Klassenentwurf

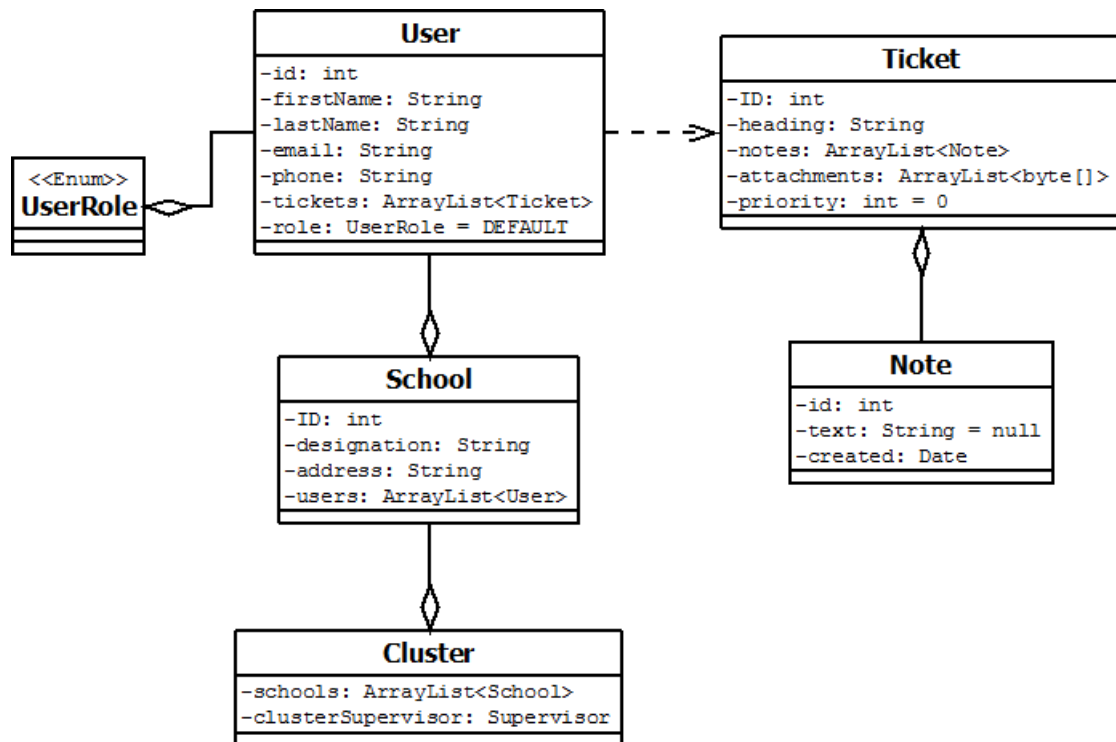


Abbildung 4.1.: Klassenentwurf Java EE Ticketsystem

In Abbildung 4.1 ist ein simpler Entwurf der Model-Klassen des Ticketsystems als Klassendiagramm dargestellt. Der Benutzer (User) hält Informationen wie deren Namen, Kontaktdetails und eine Liste der vom User erstellten Tickets für die folgenden Rollen:

**IT Manager bzw. Managerinnen (Managers):** Eine Person an Tirols Schulen, verantwortlich für die lokale Infrastrukturbetreuung und Fehlerreporting

**Systembetreuer bzw. Systembetreuerinnen (Supervisors):** Bearbeitet Meldungen (Tickets) für einen Schulcluster

**Administratoren bzw. Administratorinnen (Admins):** Eine Person des Landesschulrates, verantwortlich für das Gesamtsystem (Helmut Hammerl).

Damit steht die Klasse User im Zentrum des Systems. Die Ticket-Klasse (oder eine Entität des Typs Ticket) hat eine Überschrift, eine Priorität, welche vom User (meist IT Manager(in)) definiert wird, eine Liste an Notes (Fehlerbeschreibungen, Nachrichten in Blogpost-Form) und eine Menge an Dateianhängen.

Zu Codesequenzen:

- kurze Codesequenzen direkt im Text (mit Zeilnnummern auf die man in der Beschreibung verweisen kann)
- lange Codesequenzen in den Anhang (mit Zeilennummer) und darauf verweisen (wie z.B. hier ??)

## **4.5. Akzeptanztests**

## **5. Projektevaluation**

### **5.1. Einführung**

Die Ziele und Anforderungen in unserem Projekt Communicational haben sich im Laufe des Projekts sehr stark verändert.

Zu Beginn sollte ein bestehendes Ticketsystem des Landesschulrates angepasst und optimiert werden, indem Funktionen ausgenommen und die Weboberfläche reduziert wird. Im Zuge der Projektdurchführung, im Besonderen der Evaluation des bestehenden Systems, wurde jedoch der Schluss gezogen, dass dies nicht ohne weiteres möglich ist. Durch die hohe Komplexität des Ursprungssystems sowie durch den Verzicht auf Dokumentation und objektorientierte Entwurfsmuster wurde der Aufwand für das Diplomprojekt, spät in der Projektdurchführung, als zu groß bemessen.

### **5.2. Planungsabweichungen**

Das Projektziel wurde auf Grund des geschilderten Übermaßes an zeitlichem und personellem Aufwandes abgewandelt. Nun galt es, nicht wie ursprünglich geplant die Erweiterung bzw. Anpassung des bestehenden, auf OSTicket basierenden Systems des Landesschulrates Tirol durchzuführen. Es galt nun, einen Lösungsansatz für ein kompaktes, unkompliziertes Ticketsystem zu finden, das auf Webtechnologien setzt. IT-ManagerInnen an Tirols Schulen sollen IT-Infrastrukturprobleme einfach und schnell an die zuständigen SystemadministratorInnen bekannt geben können.

### 5.2.1. Projektpartner & -betreuer

Die Abänderungen im Projekt wurden mit dem Projektpartner Herrn OStR. Prof. Mag. Helmut Hammerl und dem Projektbetreuer Herrn Stefan Stolz, Msc beschlossen. Beide waren nach Präsentation der Zwischenergebnisse (der hohe Arbeitsaufwand, der mit dem ursprünglichen Projektziel einher gegangen wäre) der Meinung, das Ziel des Projektes in eine akademischere Richtung zu lenken.

## 5.3. Zusammenarbeit

### 5.3.1. Arbeitsaufteilung Projektteam

Die Änderungen im Projektscope zeigen eine starke Abweichung der Aufgabenbereiche der einzelnen Projektmitglieder.

Die neu zugewiesenen, individuellen Projekteinhalte:

- **Jakob Tomasi:** Das erstellen eines Systementwurfs um eine Alternative für OSTicket zu bieten und der Entwurf eines JavaEE Prototypen.
- **Peter Pollheimer:** Die Evaluierung von OSTicket und weiteren Alternativen wie OSTicky und Katak.
- **Elias Gabl:** Datenbankmodellierung und Systementwurf für die Alternative zu OSTicket.

Die ursprünglichen, individuellen Projekteinhalte:

- **Jakob Tomasi:** Das Einlesen in OSTicket's Quellcode und die Implementierungen der gewünschten Änderungen im Backend
- **Peter Pollheimer:** Die Evaluierung von OSTicket und Planung/Design/Erstellung der neuen Weboberfläche
- **Elias Gabl:** Datenbankmodellierung, -Planung, und -Änderung von OSTicket.



## 6. Zusammenfassung

- Etwas längere Form des Abstracts
- Detaillierte Beschreibung des Outputs der Arbeit

# Abbildungsverzeichnis

1.1. Risikomatrix . . . . .	13
1.2. IST-Zustand OS-Ticket . . . . .	14
2.1. OSTicket Logo . . . . .	28
2.2. Die standard-Weboberfläche für Administratoren . . . . .	29
2.3. Twitter Bootstrap Logo . . . . .	30
2.4. Bootstrap ist ein vielseitiges und anpassbares Framework . . . . .	30
2.5. PHP Ver. 7 Logo . . . . .	31
2.6. MySQL Logo . . . . .	32
2.7. Netbeans User Interface . . . . .	34
2.8. Netbeans IDE Logo . . . . .	34
2.9. XAMPP Control Panel . . . . .	35
2.10. XAMPP Logo . . . . .	36
2.11. MySQL Workbench Start Screen . . . . .	36
2.12. MySQL Workbench - Datenbankdiagramm (Ausschnitt) . . . . .	37
2.13. Komprimiertes Klassendiagramm . . . . .	38
2.14. Ticket erstellen . . . . .	39
2.15. Ticketverwaltung . . . . .	40
2.16. EER-Diagramm-Java-EE-Anwendung . . . . .	66
3.1. Mockup Anwendersicht . . . . .	85
3.2. Mockup Systembetreuer . . . . .	86
3.3. Mockup Ticketstatus . . . . .	87
3.4. Prototyp Ticketstatus . . . . .	87
4.1. Klassenentwurf Java EE Ticketsystem . . . . .	93

# Tabellenverzeichnis

1.1. Analyse Einwirkung & Auswirkung . . . . .	12
1.2. Stakeholder Identifikation . . . . .	17
1.3. Stakeholder Klassifikation . . . . .	17
2.1. Verwendete Dateitypen in OSTicket . . . . .	21
2.2. tab:ost-config . . . . .	42
2.3. tab:ost-attachment . . . . .	43
2.4. tab:ost-canned-response . . . . .	44
2.5. tab:ost-content . . . . .	45
2.6. tab:ost-department . . . . .	46
2.7. tab:ost-department2 . . . . .	47
2.8. tab:ost-faq . . . . .	47
2.9. tab:ost-staff . . . . .	48
2.10. tab:ost-staff2 . . . . .	49
2.11. tab:ost-staff3 . . . . .	50
2.12. tab:ost-ticket . . . . .	51
2.13. tab:ost-ticket2 . . . . .	52
2.14. tab:ost-user . . . . .	53
2.15. tab:ost-user-account . . . . .	54
2.16. tab:ost-ticket-priotity . . . . .	55
2.17. tab:ost-help-topic . . . . .	56
2.18. tab:ost-help-topic2 . . . . .	57
2.19. tab:tpl-user . . . . .	67
2.20. tab:tpl-ticket . . . . .	68
2.21. tab:tpl-note . . . . .	69
2.22. tab:tpl-role . . . . .	70
2.23. tab:tpl-school . . . . .	70

2.24. tab:tpl-cluster . . . . .	71
3.1. Use-Case C00 . . . . .	77
3.2. Use-Case C01 . . . . .	78
3.3. Use-Case C02 . . . . .	79
3.4. Use-Case C03 . . . . .	80
3.5. Use-Case C04 . . . . .	80
3.6. Use-Case C05 . . . . .	81
3.7. Use-Case C06 . . . . .	81
3.8. Use-Case C07 . . . . .	82
3.9. Use-Case C08 . . . . .	83

# Quelltexte

2.1. main.inc.php . . . . .	23
2.2. pwreset.php . . . . .	24
2.3. offline.php . . . . .	25
2.4. client.inc.php . . . . .	25
2.5. mysqli.php/function-db_connect1 . . . . .	59
2.6. mysqli.php/function-db_connect2 . . . . .	60
2.7. mysqli.php/function-db_connect3 . . . . .	61
2.8. mysqli.php/function-db_connect4 . . . . .	62
2.9. mysqli.php/function-db_query1 . . . . .	63
2.10. mysqli.php/function-db_query2 . . . . .	64
2.11. mysqli.php/function-db_query3 . . . . .	65
2.12. Datenbankbindung.java/Datenfelder . . . . .	72
2.13. Datenbankbindung.java/Methode-dbconnect . . . . .	73
2.14. Datenbankbindung.java/Methode-dbconnect . . . . .	74
2.15. Datenbankbindung.java/Methode-getRoles . . . . .	76
4.1. List.xhtml . . . . .	90

# Literaturverzeichnis

[Schiesser und Schmollinger 2015] SCHIESSER, Markus ; SCHMOLLINGER, Martin:  
*Workshop Java EE 7: Ein praktischer Einstieg in die Java Enterprise Edition mit dem Web Profile.* dpunkt.verlag GmbH, 2015

[Stephenson 1999] STEPHENSON, Neal: *In the Beginning ... Was the Command Line.* William Morrow Paperbacks, 1999. – URL  
[http://www.ebook.de/de/product/1695484/neal\\_stephenson\\_in\\_the\\_beginning\\_was\\_the\\_command\\_line.html](http://www.ebook.de/de/product/1695484/neal_stephenson_in_the_beginning_was_the_command_line.html). – ISBN 0380815931

# **A. Anhang-Kapitel**

## **A.1. Anhang-Section**

Testtext